Wolfgang Huber

Entwurf eines Software Asset Management (SAM) für die Niederösterreichische Landesverwaltung

DIPLOMARBEIT

HOCHSCHULE MITTWEIDA

UNIVERSITY OF APPLIED SCIENCES

Fakultät Elektro- und Informationstechnik

Mittweida, 2010

Bibliographische Beschreibung:

Huber, Wolfgang:

Entwurf eines Software Asset Management (SAM) für die Niederösterreichische Landesverwaltung.

Veröffentlichungsjahr: 2010
Seitenanzahl: 111

Diplomarbeit an der Fakultät Elektro– und Informationstechnik, der Hochschule Mittweida.

Referat:

Ziel der Diplomarbeit ist es, einen Entwurf für ein Software Asset Management für die Niederösterreichische Landesverwaltung zu erarbeiten. Das zu entwerfende Software Asset Management soll die Empfehlungen des de facto Standards ITIL berücksichtigen.

Es werden die Anforderungen spezifiziert und eine Marktanalyse bestehender Softwareprodukte durchgeführt. Die Ergebnisse der Marktanalyse werden der Möglichkeit einer Eigenentwicklung gegenübergestellt und die Entscheidung für die Eigenentwicklung dargestellt. Anhand eines Prototyps wird das entworfene Systemkonzept verifiziert. Es werden die eingesetzten Technologien beschrieben, insbesondere die Eignung und Möglichkeiten der Windows Presentation Foundation für datenorientierte Anwendungen werden erläutert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

AD	Active Directory
API	Application Programming Interface
BU	Abt. LAD1-IT Fachbereich Benutzerunterstützung
COBIT	Control Objectives for Information and related Technology
DDL	Data Definition Language
FTP	File Transfer Protocol
GV	Abt. LAD1-IT Stabstelle Geräteverwaltung
HTTP	Hyper Text Transfer Protocol
IANA	Internet Assigned Numbers Authority
ISO	International Organization for Standardization
IT	Information technology
ITIL	IT Infrastructure Library
LAD1-IT	Abteilung Landesamtsdirektion/Informationstechnologie
LAKIS	Landeskommunikations- und Informationssystem
NÖL	Amt der Niederösterreichischen Landesregierung
OU	Organisational Unit
PC	Personal Computer
SAM	Software Asset Management
SCCM	Microsoft System Center Configuration Manager
SIT	Software Information Tag
SQL	Structured Query Language
UML	Unified Modeling Language
WinSock	Windows Sockets
WMI	Windows Management Instrumentation
WPF	Windows Presentation Foundation
XAML	Extensible Application Markup Language
XML	Extensible Markup Language

1 Einleitung

1.1 Motivation

Software ist eines der wichtigsten Assets[1] ähnlich der physischen Vermögensgegenstände wie PCs oder Server in einem Unternehmen. Da Software in den meisten Unternehmen einen beträchtlichen Vermögensgegenstand darstellt, und beim Erwerb die Lizenzbestimmungen der Hersteller akzeptiert werden müssen, ist die korrekte Inventarisierung unabdingbar.

Zurzeit werden in der Niederösterreichische Landesverwaltung für die Aufzeichnung der installierten bzw. zugeteilten Softwareprodukte unterschiedliche Systeme eingesetzt. Dies reicht vom Einsatz eines kommerziellen Systems bis zum Führen einfacher Excel-Listen in einzelnen Fachabteilungen. Als Datenbasis dient allen Systemen aber der Soll-Zustand der aus den Anforderungen der Anwender resultiert. Ein Abgleich des eingetragenen Soll-Zustandes mit dem Ist-Zustand ist mit diesen Lösungen nicht möglich.

Ziel ist es, ein System zu schaffen, das die benötigten Funktionen beinhaltet, um eine nachvollziehbare Inventarisierung durchführen zu können. Die Funktionen sollen dabei, soweit wie möglich, in die bestehenden IT-Prozesse wie auch in die IT-Infrastruktur integriert werden.

[1] Asset: Begriff für Vermögensgegenstand

1.2 Gliederung

Kapitel 2 gibt einen Überblick über die Niederösterreichische Landesverwaltung und eine kurze Beschreibung des im Einsatz befindlichen Asset Management

Im **Kapitel 3** werden die Funktionen eines gesamtheitlichen Software Asset Management anhand der aktuellen Standards erläutert. Es wird eine Marktanalyse von möglichen Softwareprodukten durchgeführt und die Kriterien für die Entwicklung werden definiert.

Das **Kapitel 4** beinhaltet den Systementwurf. Hier werden im speziellen die eingesetzten Technologien, die zum Einsatz kommen, beschrieben.

Kapitel 5 beschreibt die eigentliche Implementierung des Prototyps anhand von wesentlichen Programmausschnitten die die Grundfunktionalitäten beinhalten.

Im **Kapitel 6** werden die Ergebnisse und Erkenntnisse der Arbeit zusammengefasst und weitere mögliche Ausblicke vorgestellt.

2 Die Niederösterreichische Landesverwaltung

Dieses Kapitel beschreibt die NÖ Landesverwaltung, die Abteilung LAD1-IT und die Anwendung „RW-Assetmanagement“ und soll in groben Zügen zeigen wie das Asset Management in der der Niederösterreichischen Landesverwaltung derzeit abgebildet ist.

2.1 Amt der Niederösterreichischen Landesregierung

Das Amt der Niederösterreichischen Landesregierung (NÖL) besorgt die Geschäfte des selbständigen Wirkungsbereiches des Landes und der mittelbaren Bundesverwaltung als Hilfsorgan der NÖ Landesregierung. Das Amt der Niederösterreichischen Landesregierung gliedert sich nach Maßgabe seiner Geschäftseinteilung in Abteilungen und Gruppen. (vgl. [RIS, 2010]).
Die Agenden der Informations- und Kommunikationstechnologie sind in der Abteilung Landesamtsdirektion/Informationstechnologie (LAD1-IT) zusammengefasst.

2.2 Abteilung LAD1-IT

Die Abteilung LAD1-IT umfasst zurzeit 116 Mitarbeiter. Im Betreuungsbereich stehen 6500 Anwender, welche auf über 100 Standorte verteilt sind. Es werden sowohl die Entwicklung wie auch der Betrieb von modernen IT-Systemen zur Unterstützung der Verwaltungsaufgaben verantwortet. Die einzelnen Aufgaben sind den so genannten Fachbereichen zugeordnet. Die Fachbereiche sind Anwendungsentwicklung, Benutzerunterstützung, Betriebsführung und Planung/Integration. Zusätzlich zu den Fachbereichen sind so genannte Stabstellen installiert. Zu den Stabstellen gehören unter anderen die Geräteverwaltung, IT-Sicherheitsbeauftragter und E-Governmentbeauftragter (Organigramm der Abteilung LAD1-IT siehe Anlage 1).

Neben den Mitarbeitern der Abteilung LAD1-IT sind in den einzelnen Abteilungen, je nach Größe, ein bis zwei IT-Koordinatoren tätig, die die Anwender Vorort betreuen. In den Tätigkeitsbereich der IT-Koordinatoren fällt auch die korrekte Inventarisierung der für die eigene Abteilung zugeteilten Hardware und Software.

2.3 IT-System „RW-Assetmanagement“

Das IT-System „RW-Assetmanagement“ wurde im Jahr 2000 in der Abteilung LAD1-IT konzipiert und eingesetzt. Das System dient zur Verwaltung aller Niederösterreichweit eingesetzten IT-Komponenten (Hardware und Software) sowie von Telefonen während des gesamten Lebenszyklus (siehe Anlage 2). Die kaufmännischen Aspekte (Anforderung, Bestellung, Auslieferung) standen beim Entwurf im Vordergrund. Das Assetmanagement unterstützt die Vorgänge im Bereich Lieferung, Installation, Wartung und Störungsmeldung. Anlage 3 zeigt den Prozess „Asset-Management“ im Überblick. Das Hauptaugenmerk bei der Einführung lag ursprünglich bei der Verwaltung der Hardware Assets der Niederösterreichischen Landesverwaltung. Abbildung 1 zeigt einen Anzeigebildschirm für ein Asset mit den verknüpften Komponenten.

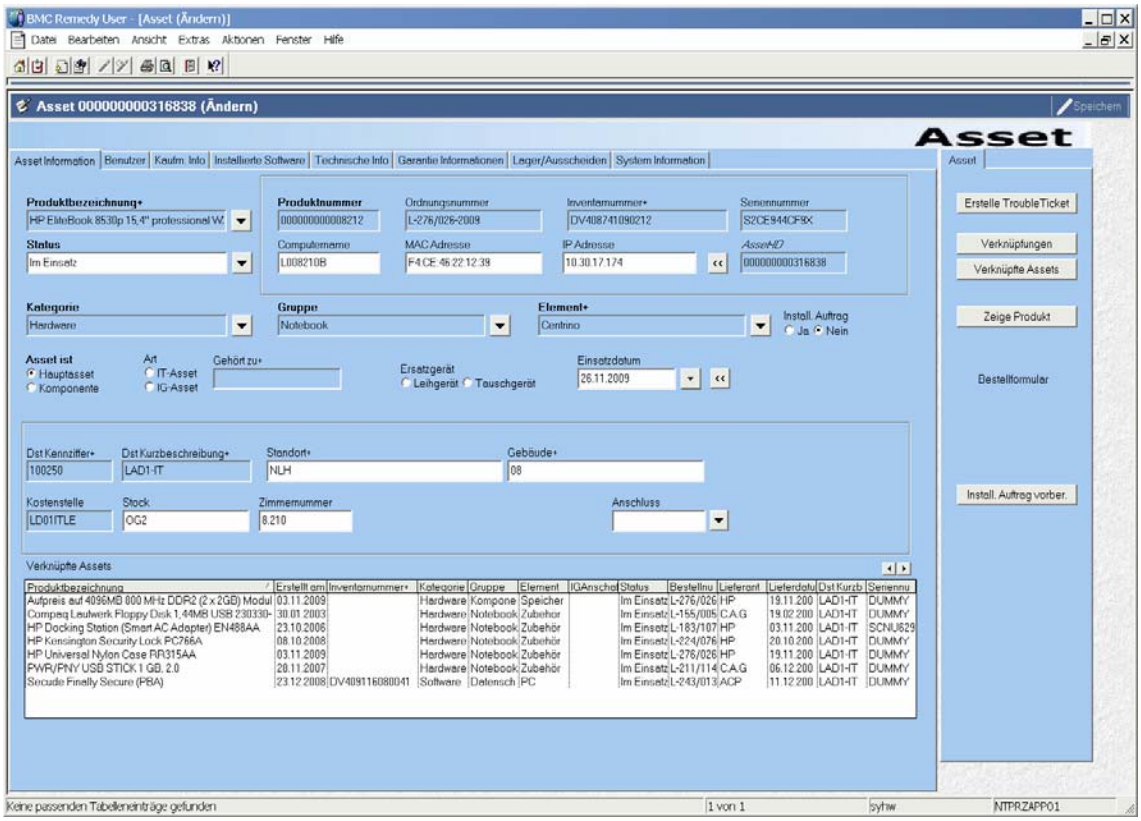


Abbildung 1: Anzeigebildschirm „Asset“ im RW-Assetmanagement

Das System ist mit dem Produkt Remedy der Firma BMC realisiert. Die Anwendung ist speziell auf die Bedürfnisse der LAD1-IT adaptiert worden.

Zusätzlich zu der Verwaltung von Hardware Assets wurden noch einfache Erfassungsmasken für Software, Lizenz und Vertragsmanagement vorgesehen. Hierbei handelt es sich aber in der Regel um die Erfassung nicht strukturierter Daten. Das zugrunde liegende Datenbankmodell (siehe Anlage 4) ist grundsätzlich offen. Da es aber weitestgehend über die Eingabemasken generiert und Abhängigkeiten über die Programmlogik abgedeckt werden ist nur ein lesender Zugriff sinnvoll. Für den lesenden Zugriff werden so genannte Views zur Verfügung gestellt. Der Vorteil liegt darin, dass hier auch sprechende Feldnamen vergeben werden können. Remedy selbst nummeriert Informationsfelder einfach durch.

2.3.1 Überblick über die generelle Funktionalität

Folgende Hauptfunktionalitäten wurden implementiert (vgl. [Soltys, 2000] S. 31):

- Bestellung und Lieferungsbearbeitung,
- Installationsbearbeitung,
- Störungsbearbeitung,
- Wartungsvertragsbearbeitung,
- Stammdatenverwaltung,
- Angebotsbearbeitung,
- Flexibles Erstellen von Datenbankabfragen und Reports,
- Zentrale Verwaltung der Assets,
- Dokumentation sämtlicher Statusänderungen.

2.3.2 Beschreibung der Kommunikationswege

Über Anforderungen oder Problemmeldungen (Störungen) werden Änderungsprozesse an den IT-Arbeitsplatzausstattungen ausgelöst. Hierbei werden Änderungen an vorhandenen Assets durchgeführt oder neue beschafft. Die zentrale Verwaltung der Assets obliegt der Stabsstelle Geräteverwaltung (GV) und dem Fachbereich Benutzerunterstützung (BU). Die Kommunikationswege dieser Änderungsprozesse sind eindeutig definiert und in Abbildung 2 dargestellt.

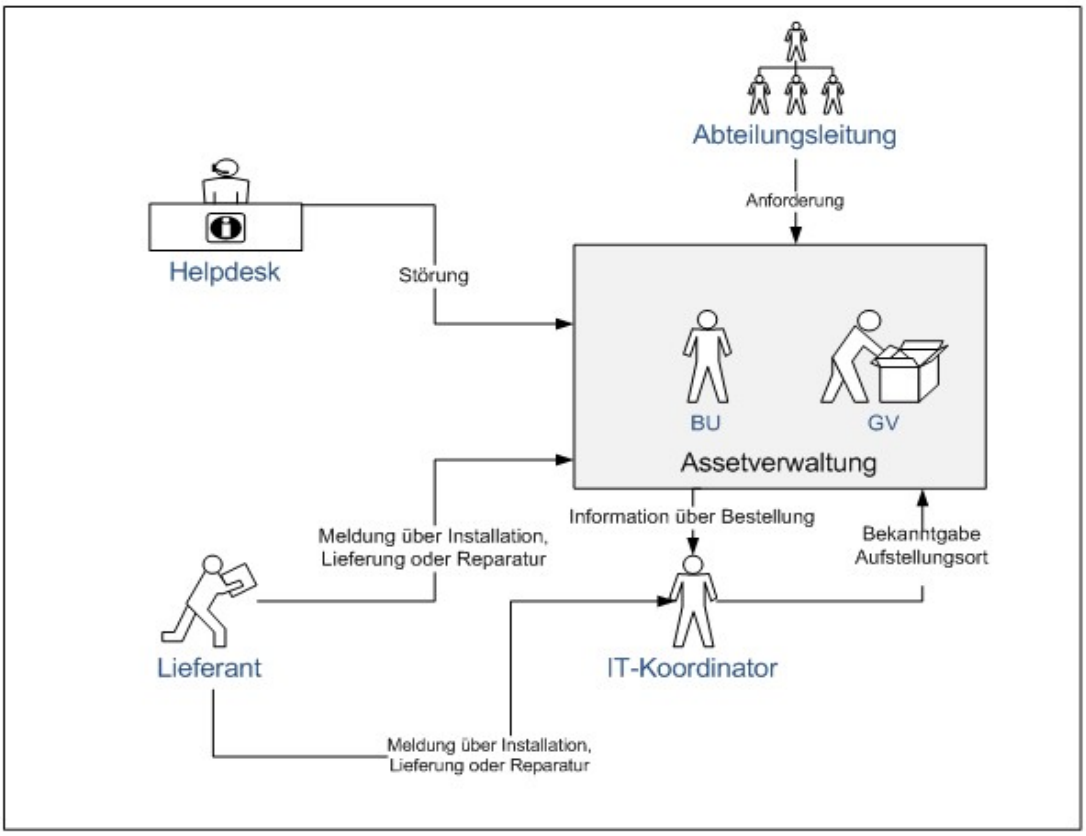


Abbildung 2: Kommunikationswege

2.3.3 Rollen

Um die Aufgabenverteilung zu verdeutlichen werden die zentralen Verantworlichkeiten beschrieben.

Helpdesk

Zweck	Stößt durch eine Anfrage/Problemmeldung (Troubleticket) einen Beschaffungs- oder Reparaturprozess an und erhält das angeforderte Asset innerhalb der vorgegebenen Lieferzeit. Lesender Zugriff auf das System
Verantwortung	1. Beschaffung auslösen bzw. anregen 2. Assetinformationen nutzen 3. Auslösen einer Reparatur durch ein Troubleticket
Rollenbesetzung	Jeder Helpdesk-Mitarbeiter

Abteilungsleitung

Zweck	Stößt durch eine Anforderung im LAKIS einen Beschaffungsprozess an.
Verantwortung	Beschaffung auslösen bzw. anregen
Rollenbesetzung	Gesamte Verwaltung

Lieferant

Zweck	Lieferung und Reparatur defekter Assets
Verantwortung	Entgegennahme von Bestellungen Vorbereitung und Installation von bestellten Assets Lieferung von bestellten Assets Reparatur defekter Assets Bereitstellung funktionsfähiger Assets
Rollenbesetzung	Externe Lieferanten, die für die Lieferung oder Reparatur von Assets verantwortlich sind.

Geräteverwaltung (GV)

Zweck	IT-mäßige Verwaltung der Assets. Schreibender Zugriff auf das System
Verantwortung	Einholen von Angeboten Übernahme von Lieferungen Anlegen von (bestellten) Assets im Assetmanagement Änderungen im Asset-Management nachziehen Ergänzung fehlender Informationen im Asset-Managementsystem Kontrolle der gelieferten Assets Installation (gemeinsam mit BU) der Assets veranlassen Reparatur von beschädigten Assets veranlassen (ext. Lieferant) Gerätetausch im Störungsfall Ausscheiden der Assets im System gewährleisten Stammdatenverwaltung Ggf. Wartungsverträge verwalten
Rollenbesetzung	Mitarbeiter der Geräteverwaltung (GV)

Benutzerunterstützung (BU)

Zweck	Durchführen von Arbeiten, die die Assets betreffen
Verantwortung	Generelles Arbeiten mit Assets (Installation, Deinstallation etc.) Zuweisung von Geräten an Anschlussdosen Aufstellen Installieren von Geräten/Assets Aktualisieren der Daten im Assetmanagement
Rollenbesetzung	Mitarbeiter der Benutzerunterstützung (BU)

IT-Koordinator

Zweck	IT-Koordinatoren sind für die Assets ihrer Dienststelle verantwortlich
Verantwortung	Instandhaltung der in seinem Bereich befindlichen Assets Installation bzw. Anwesenheit bei Lieferungen Zuweisung von Assets zu den Arbeitsplätzen und Usern Vergabe von IP-Adressen Aktualisieren der Daten im RW-Assetmanagement (Zugriff auf Assets die seinem Bereich zugeordnet sind)
Rollenbesetzung	Definierte Mitarbeiter der NÖL in den Abteilungen

3 Software Asset Mangement

3.1 Definition

Die Problematik, dass Software durch ihre digitale Natur nicht greifbar ist erschwert die Inventarisierung und Nachvollziehung der vorhandenen Software und ihrer Versionsstände. Das OGC[2] beschreibt Software Asset Management (SAM) als Integration der notwendigen Prozesse und Infrastruktur (IT-System) um Software zu managen, nachzuverfolgen und eine korrekte Lizenzierung zu gewährleisten (vgl. [Rudd, 2009], S. 23ff). Die Art der Lizenzierung hängt von Hersteller ab und kann auch durchaus differieren auf welcher Hardwareplattform sie zum Einsatz kommt. Hier ist vor allem Serversoftware zu erwähnen, welche auf Basis von z.B.: Anzahl der Prozessoren lizenziert werden muss. Daraus ergibt sich, dass eine SAM-Lösung auch die für die Lizenzierung notwendige Hardwarekonfiguration, auf der die Software installiert ist, kennen muss. Da Software letztlich einen Vermögenswert darstellt ist eine Inventur, wie bei anderen Werten wie Dienstwagen, Maschinen unbedingt erforderlich (vgl. [Marco, 2006]).
Abbildung 3 zeigt den durch die ISO/IEC 19770 definierten SAM-Prozess. Es werden auch die Schnittstellen zu den ITIL-Prozessen dargestellt. Die Darstellung zeigt, dass neben den eigentlichen Kernaufgaben auch der komplette Lebenszyklus eines Prozesses betrachtet werden soll. Auf die Rahmenprozesse, welche die Kernprozesse begleiten, wird in dieser Arbeit nicht eingegangen.

[2] OGC: Office of Government Commerce

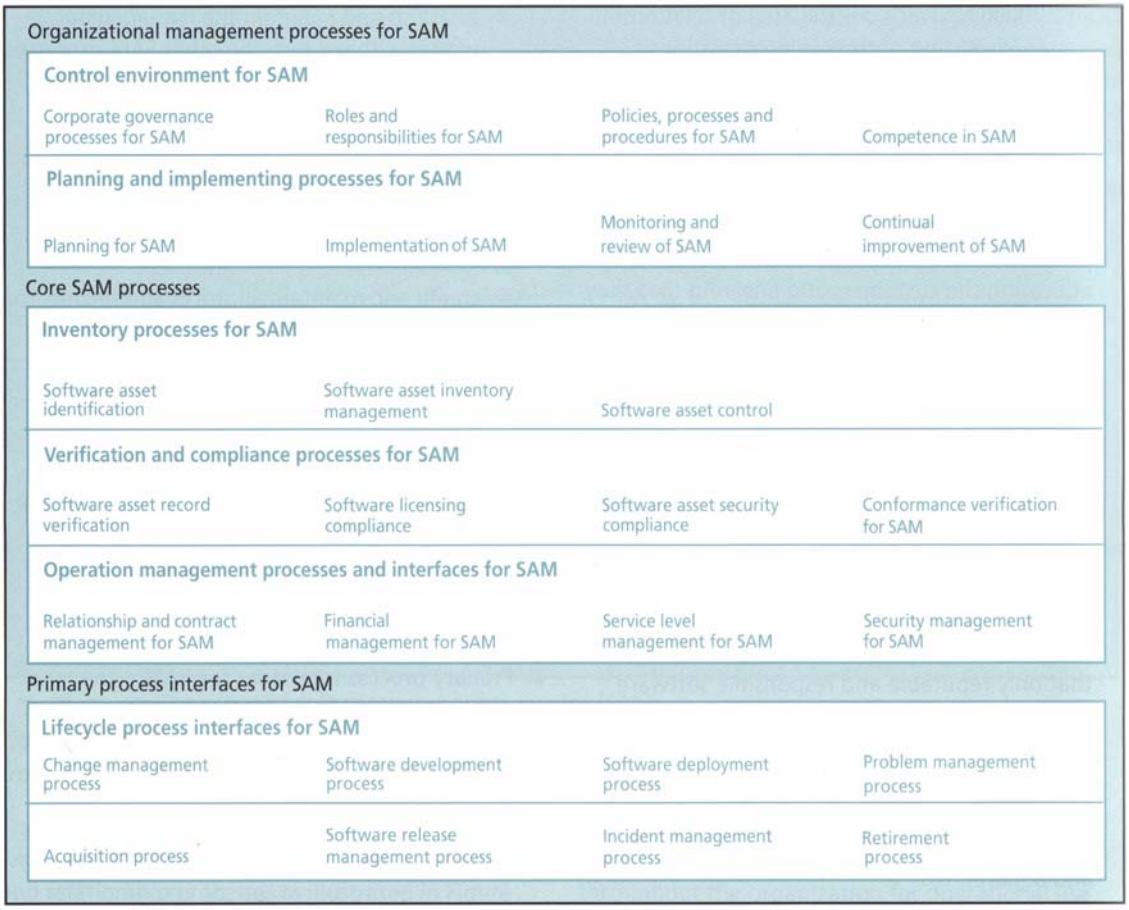


Abbildung 3: ISO/IEC 19770 SAM Prozess [3]

Auch die Darstellung (siehe Abbildung 4) in ITIL (definiert durch die OGC) lehnt sich stark an die Definition in der ISO/IEC 19770 an und umreist die gleichen Teilprozesse bzw. Themenbereiche wenn auch in unterschiedlicher Gruppierung.

[3] Bildquelle: [Ruud, 2009] S.104

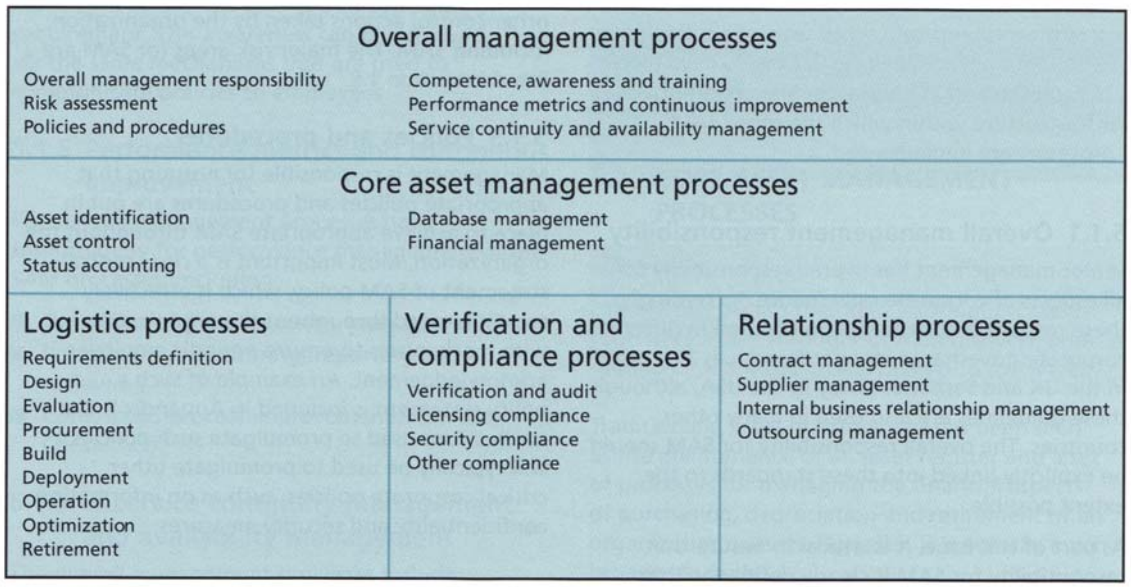


Abbildung 4: SAM Prozessbereiche nach ITIL[4]

Um Software Asset Management nach ISO/IEC 19770 wie auch ITIL ([5], S. 50ff.) betreiben zu können müssen folgende Aufgabenbereiche abgedeckt werden:

- Asset Management Prozess,
- Logistik Prozess,
- Verifikation und Compliance Prozess.

3.1.1 Asset Management Prozess

Jede Organisation muss als erstes definieren, welche Assets in das SAM-System aufgenommen werden müssen. Zu unterscheiden ist noch, ob die Assets aus Inventarisierungssicht aufgenommen werden oder zusätzlich auch durch das SAM-System über Ihren Lebenszyklus gesteuert werden soll. Im Allgemeinen müssen folgende Assets durch das SAM-System gesteuert werden:

- Erworbene Lizenzen (Kauf, Miete, etc.),
- Vertragsdokumentation, Lizenzbestimmungen,
- Installierte Software,

[4] Bildquelle: [Ruud, 2009] S. 47

- Finanzdaten (z.B.: zum Abrechnen bzw. Zuteilung von Kosten auf Kostenstellen),
- Definitionen von Standardkonfigurationen (z.B.: Standard-Client der Organisation).

Weiters muss dafür Sorge getragen werden, dass die Assets autorisiert über den Lebenszyklus zum Einsatz kommen. Die Dokumentation des Status, wie z.B.: installiert oder ausgeschieden, und die Nachvollziehbarkeit der Änderungen ist ebenfalls notwendig.
Als Letztes beinhaltet dieser Prozess noch die Erfassung und Auswertung der Kosten. Das umfasst sowohl die Kosten für ein Software Asset von der Beschaffung über die Wartung bis zum Ausscheiden wie auch die Zuordnung zu betrieblichen Kennzahlen (Zuordnung zu Kostenstellen).
Der Assement Management Prozess stellt somit den Soll-Zustand der Software Assets in einer Organisation dar.

3.1.2 Logistische Prozesse

Der Logistik Prozess umfasst alle Aktivitäten rund um den Lebenszyklus eines Software Assets. Wie in Abbildung 5 dargestellt wird von der Evaluierung (Evaluation) über die Beschaffung (Procurement) bis zum Ausscheiden (Retirement) der Lebenszyklus beschrieben.

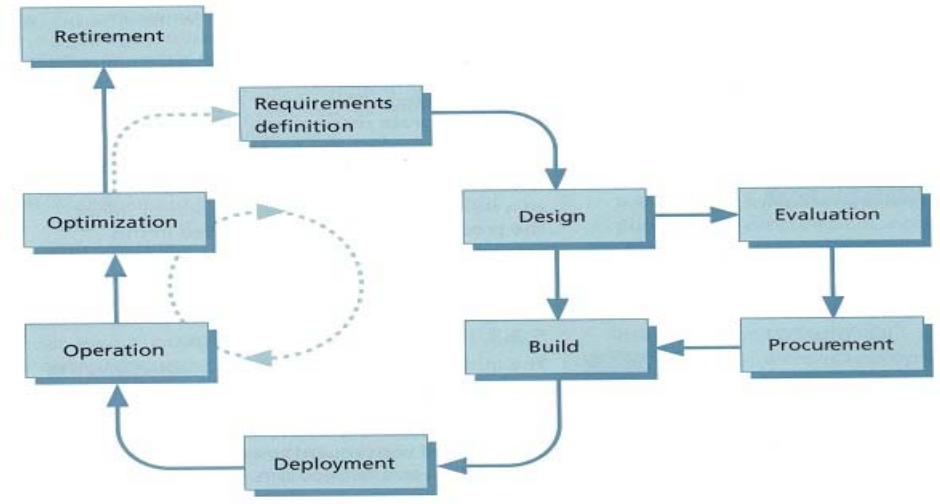


Abbildung 5: Lebenszyklus eines Software Assets[5]

[5] Bildquelle: [Ruud, 2009] S. 52

Der Prozess soll sicherstellen, dass die passende Software am passenden Ort mit der entsprechenden Qualität zur Verfügung gestellt wird. Darum steht am Anfang einer jeglichen Softwareanforderung die Anforderungsdefinition (Requirements definition) die sowohl die funktionalen Anforderungen wie auch Anforderungen an die Benutzbarkeit und auch nicht funktionale Anforderungen wie Unterstützung und Betrieb der Software beinhaltet. Bei einer Eigenentwicklung werden die Schritte Entwurf (Design) und Erstellung (Build) bei Zukauf einer externen Software werden zusätzlich die Schritte Bewertung (Evaluation) und Beschaffung (Procurement) durchlaufen. Danach folgen die Schritte für die Produktivsetzung (Deployment) und der eigentliche Betrieb (Operation). Am Ende des Lebenszyklus eines jeglichen Software-Assets steht das ordnungsgemäße Ausscheiden (Retirement). Die Schritte des Logistikprozesses sind meist in den Softwareerstellungsprozessen der Organisationen abgebildet.

3.1.3 Verifikation und Einhaltung

Der technisch wie auch organisatorisch anspruchsvollste Prozess beinhaltet das Vorgehen zum Soll-Ist-Vergleich der Software-Assets. Dieser Prozess dient weiters zur Erkennung und Eskalation aller Ereignisse in Zusammenhang mit den SAM-Grundsätzen, -Prozessen und Lizenznutzungsrechten ([OGC, 2007], S.50ff.). Abbildung 6 zeigt den grundsätzlichen Ablauf und die Interaktion zwischen der realen Welt (Ist-Zustand) und der Abbildung in der SAM-Datenbank (Soll-Zustand).

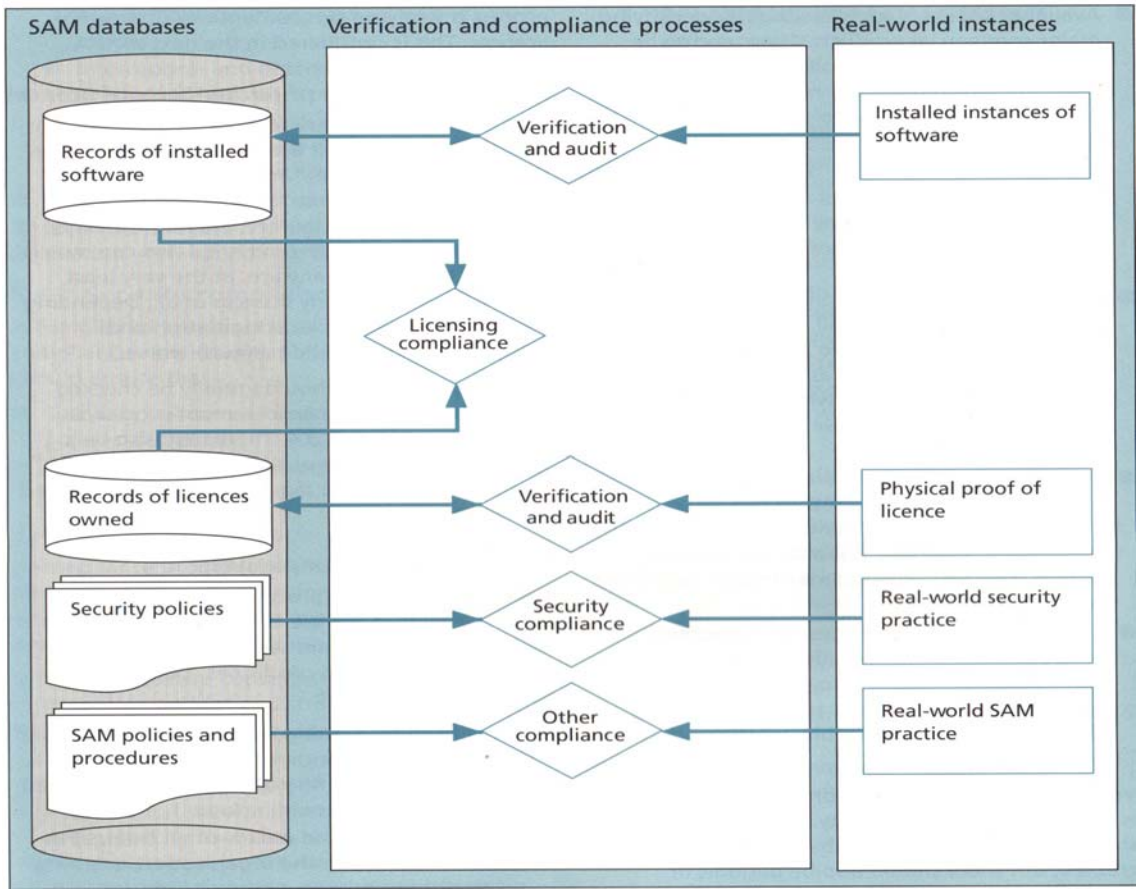


Abbildung 6: Verifikation und Einhaltung[6]

Die Hauptaufgabe in der Verifikation und Überwachung (Verification and audit) liegt darin die Richtigkeit der SAM-Datensätze (Software, Lizenzen) sicherzustellen. Als Herausforderung ist einerseits das Ermitteln der real installierten Software anderseits der automatisierte Vergleich zu den in der SAM-Datenbank eingetragenen Datensätzen zu sehen. Hier ist die eindeutige Identifikation einer Software bereits schwierig, da sich noch kein breitflächig eingesetzter Standard etabliert hat. Abhilfe kann hier der Standard IEC 19770-2 (vgl. [ISO/IEC 19770-2]) schaffen, der sich mit der eindeutigen Identifikation von Software beschäftigt.

[6] Bildquelle: [Ruud, 2009] S. 61

3.1.4 Fazit

Um die Aufgaben des Software Asset Management zu gewährleisten ist ein IT-System unabdingbar. Das IT-System unterstützt einerseits den gesamten Beschaffungsprozess (Beschaffung, Inventarisierung, Aufnahme von Lizenz- und Wartungsverträgen). Diese kaufmännischen Aspekte sind oftmals schon im Beschaffungs- oder Buchhaltungssystemen integriert. Andererseits ist aber der automatisierte Abgleich entscheidend, welcher den Ist-Zustand über die installierte Software abbildet und mit dem Soll-Zustand vergleicht. Auf diese Kernaufgabe setzen die meisten auf dem Markt befindlichen SAM-Systeme auf. Wobei der Grad der Prozessunterstützung bzw. Anpassungsfähigkeit auf organisationsspezifische Bedürfnisse unterschiedlichste Ausprägungen findet. Die Schwierigkeit liegt in der Integration der IT-Systeme da Ist-Zustand und Soll-Zustand meist in verschieden Systemen abgebildet sind und das aus organisatorischen wie kaufmännischen Gründen auch so bleiben soll.

3.2 Grundaufbau von Software Asset Management Systemen

Mit SAM-Systemen soll es möglich sein die Einhaltung der Softwarelizenzen aktuell und wiederkehrend in einem Unternehmen zu erfassen und auch zu überwachen. Die am Markt befindlichen Systeme umfassen zum Teil auch Funktionen wie die Verwaltung von Verträgen, Messung der tatsächlichen Softwarenutzung und automatische Alarmierung bei Nutzung unerwünschter Software.

3.2.1 Ableitung des Begriffes Informationssystem

[Alpar et al.,2000] S. 28 liefert folgende Definition für ein Informationssystem:

> "Ein Informationssystem ist ein künstliches, konkretes System, das aus maschinellen und natürlichen Elementen besteht und seine Nutzer mit Informationen versorgt. Es ist gleichzeitig ein Element einer Organisation oder Organisationsbeziehung."

Nach [Winter/Aier, 2010] definiert sich ein Informationssystem wie folgt:

> „Ein Informationssystem (IS) ist ein soziotechnisches System, welches die Elementtypen Aufgabe, Technik und Mensch umfassen kann (Heinrich, 1990). Ein Informationssystem beinhaltet Informationen für die Durchführungs-, Führungs-, Analyse- und Entscheidungsfunktionen einer Organisation und stellt diese den Aufgabenträgern zur Verfügung (Davis und Olson, 1985)."

Durch ihre Datenorientierung lassen sich SAM-Systeme in die Reihe von Informations- und Datenbanksysteme einordnen. Es fällt auf, dass der organisatorische Aspekt bei einem Informationssystem nicht zu vernachlässigen ist. Wendet man dies auf das zu entwerfende Informationssystem an, ergibt sich folgendes Gesamtbild:

- Ein Informationssystem besteht aus verknüpften Elementen. In Bezug auf das Informationssystem, das in dieser Arbeit entworfen werden soll, trifft dies auf die zu verknüpfenden Komponenten SAM-Datenbank, RW-Assetmanagement und Clientanwendung zu. In zweiter Linie kann man das auch auf die organisatorischen Einheiten, die hinter den Systemen stehen anwenden, denn ein Informationssystem ist wie oben erwähnt wurde, auch Teil einer Organisation oder Organisationsbeziehung.
- Ein Informationssystem hat ein Ziel, nämlich die Problemlösung (siehe Kap. 3.3). Für die Lösung sind Informationen verantwortlich, die dadurch entstehen können, dass die verknüpften Elemente miteinander interagieren. Im Kontext der Diplomarbeit ist das Hauptziel die Verifikation und Einhaltung der installierten Software mit dem erfassten Zustand im RW-Assetmanagement. Darüber hinaus ist es Ziel, diese Informationen mit weiteren passenden Informationen aufzuwerten, die sich durch die Verknüpfung der einzelnen Komponenten ergeben.

3.2.2 Architektur eines Informations- und Datenbanksystem

Datenorientierte Systeme beschäftigen sich nicht nur mit der Speicherung der Daten sondern dienen vorwiegend zur Befriedigung von Informationsbedürfnissen. Es wird daher auch häufig in diesen Zusammenhang nur mehr von Informationssystemen gesprochen. Ein wesentlicher Bestandteil ist die einheitliche Speicherung der Daten in ein Datenbankmanagementsystem. Zusätzlich stehen Funktionen zum Verarbeiten und Abfragen der Daten zur Verfügung (vgl. [Dumke, 2003] S.245).
Vereinfacht kann die Architektur eines Informationssystems wie in Abbildung 7 dargestellt werden.

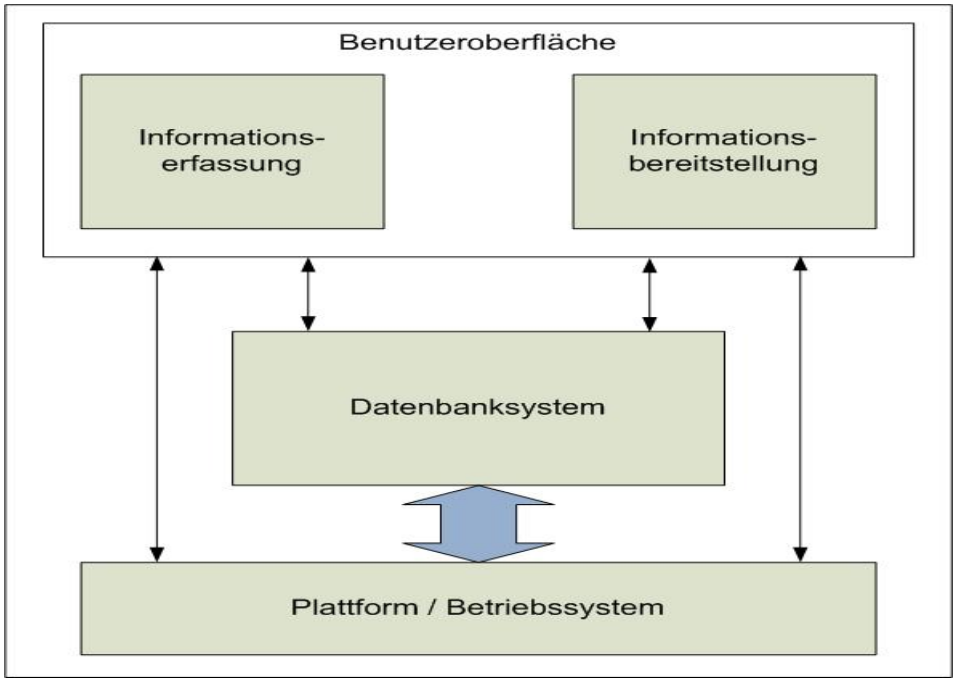


Abbildung 7: Aufbau eines Informationssystem

Die am Markt befindlichen SAM-Lösungen sind nach der Architektur eines Informationssystems aufgebaut. Sie bestehen meist aus 3 Komponenten. Die erste Komponente ist ein Discoveryprogramm welches auf den Clients installiert werden muss. Dieses Programm analysiert die Hardware und die darauf installierten Softwareprodukte. Die Erfassung der Hardware ist wie in 3.1 beschrieben wesentlich, da die Lizenzierung von der Hardware abhängig sein kann. Weiters kann in diesem Programm auch der Grad der Nutzung ermittelt werden. Das Discoveryprogramm dient zur Informationserfassung, der Anwender merkt in der Regel nichts davon beziehungsweise ist keine Interaktion nötig. Nach dem Ermitteln der relevanten Informationen werden die Daten, in definierbaren Intervallen, an eine zentrale Stelle gesendet. Die zweite (zentrale) Komponente ist zumeist auf einem dezidierten Server installiert und dient zum Empfang, Verarbeiten und Aufbereiten der Daten welche von den Clients gesendet werden. Zusätzlich zu der Verarbeitungslogik ist eine Anwendung (als Web oder Client/Serveranwendung) vorhanden, die das Auswerten der Daten nach verschiedensten Kriterien ermöglicht (vgl. [Nieder, 2003] S. 430ff, [Kalaher, 2010]). Die dritte Komponente bildet das Datenbankmanagementsystem. Je nach Lösung kommt zur Speicherung der Daten ein eigenes oder von einem Dritthersteller beigestelltes Datenbankmanagementsystem zum Einsatz.

3.3 Anforderungen an das Software Asset Management

Bei der Niederösterreichischen Landesverwaltung (NÖL) sind Teile des Software Asset Management Prozess etabliert. Dieser beinhaltet die Inventarisierung der lizenzierten Software und die händische Zuteilung der Lizenzen zu den Benutzern bzw. Clients. Diese Daten werden in das Produkt BMC ARS Remedy eingetragen. Anlage 4 zeigt das ER-Diagramm[7] des bestehenden Assetmanagements. Der wesentliche Bestandteil, nämlich der automatisierte Soll-Ist-Vergleich fehlt aber.

Die zu entwerfende Lösung soll das bestehende Informationssystem der reinen Inventarisierung dahingehend erweitern, dass Software Asset Management wie in Kapitel 3.1 beschrieben möglich ist.

Bei der Beschreibung der Anforderungen an ein zu entwickelndes Software-System ist es hilfreich, eine Einordnung der Problemdefinition/Anforderung zu treffen.
Nach [Dumke, 2003] S. 25 ist die Problemdefinition wie folgt definiert:

> „Die **Problemdefinition** (problem definition) ist die zusammenfassende Beschreibung von Anforderungen (requirements) an die Entwicklung von Software-Produkten die sich unterteilen in **funktionale Anforderungen** zur Beschreibung der Arbeitsweise und der grundlegenden Eigenschaften der Problembezogenen Funktionalität, **qualitative Anforderungen** zur Darstellung der Produkt-Qualität in seinen verschiedenen Ausprägungen, **Systembezogene Anforderungen** für die Charakterisierung der erforderlichen Hardware und damit verbundenen bzw. notwendigen Software und **Prozessbezogene Anforderungen** zur Beschreibung der projektspezifischen Merkmale, wie Entwicklungszeit, Kontrollpunkten personelle und finanzielle Ressourcen."

In dieser Arbeit wird die Einordnung der Anforderungen nach [Dumke, 2003] S.25ff übernommen.

3.3.1 Funktionale Anforderungsmerkmale

3.3.1.1 Clientanwendung

- Rollenkonzept: Das IT-System muss basierend auf Rollen folgende Zugriffsberechtigungen unterstützen:
  - Rolle „*Benutzer*" darf die Daten des Clients sehen auf dem er eingestiegen ist,
  - Rolle „*IT-Koordinator*" darf alle Daten des von ihm betreuten Bereiches verwalten,
  - Rolle „*Administrator*" darf alle Daten verwalten.

[7] Entity-Relationship-Diagramm

- Visualisierung: Die Daten die in der Datenbank abgespeichert sind, müssen abgefragt werden können.

3.3.1.2 Clientscanprogramm

- Ressourcenverbrauch: Der Scanvorgang muss ohne Beeinträchtigung der Arbeitsgeschwindigkeit des zu untersuchenden Gerätes durchgeführt werden können. Die Datenübertragung zum zentralen Server muss auch für schmalbandige Anbindungen geeignet sein. Weiters sollte eine Möglichkeit der händischen Datenübermittlung (für Mitarbeiter die vorwiegend im Außendienst tätig sind) vorgesehen werden.
- Änderbarkeit: Es soll möglich sein, zusätzliche Informationen zu ermitteln und an den Server zu übermitteln.
- Transparenz: Die Daten die das Clientscanprogramm ermittelt sollen dem Benutzer transparent zur Verfügung stehen. Das bedeutet, dass die übermittelten Daten, wenn gewünscht, lokal am Client abgelegt werden müssen.

3.3.1.3 Serveranwendung

- Transparenz: Die Rohdaten der empfangenen Clients müssen erhalten bleiben.
- Leistungsfähigkeit: Die Serveranwendung muss die Daten von rund 10.000 Clients zur Verarbeitung annehmen können. Die Daten müssen stundenaktuell in die Datenbank eingearbeitet werden können. Sollte die Datenverarbeitung gestoppt sein so müssen alle noch nicht importierten Daten automatisch beim Wiederanlauf verarbeitet werden.
- Interoperabilität: Es ist die Möglichkeit einer Verknüpfung zum bestehenden RW-Assetmanagement vorzusehen.
- Das Datenmodell ist so zu entwerfen, dass Auswertungen leicht möglich sind.

3.3.2 Systembezogene Anforderungsmerkmale

- Betriebssysteme: Die Clientanwendung muss als Betriebssystem Windows XP mit Service Pack 3 unterstützten.
  Die Serversoftware muss auf dem Betriebssystem Windows Server 2008 lauffähig sein. Ein Upgrade auf Folgeversionen der Betriebssysteme sollte einfach möglich sein.
- Datenbankmanagementsystem: Als Datenbankmanagementsystem ist Microsoft SQL Server 2008 einzusetzen, die bei NÖL als strategische Plattform definiert ist.

- Programmiersprachen (bei Eigenentwicklung): Als Entwicklungsplattform für die Clientanwendung ist primär Microsoft Visual Studio 2008 mit C# zu nutzen. Als Framework ist .Net Framework 3.5 oder höher einzusetzen. Anhand der Clientanwendung ist zu prüfen inwieweit die Windows Presentation Foundation (WPF) für eine datenorientierte Anwendung geeignet ist. Für die Clientscananwendung ist je nach Betriebssystem und den vorhandenen Architekturvorgaben die geeignete Programmierumgebung zu wählen.
- Architektur: Die Lösung muss sich auf der bestehenden IT-Infrastruktur der NÖL betreiben lassen. Die Serverlösung ist ausfallsicher zu gestalten. Die IT-Infrastruktur der NÖL für datenorientierte Anwendungen, in vereinfachter Form, ist in Anlage 5 dargestellt.

3.3.3 Qualitätsbezogene Anforderungsmerkmale

- Interoperabilität: Das System soll sich nahtlos in die bestehende Infrastruktur integrieren und soll bereitgestellte Funktionen nutzen.
- Sicherheit: Unberechtigter Zugriff muss verhindert werden. Jeder darf nur die Daten sehen auf die er auf Grund seiner zugeordneten Rolle berechtigt ist.
- Zeitverhalten: Datenbankzugriffe sind auf Performance hin zu optimieren. Verarbeitung vieler gleichzeitiger Zugriffe muss gewährleistet sein.
- Änderbarkeit: das System muss sich leicht an neue Anforderungen und Änderungen von Abläufen und Umgebungen anpassen lassen. Voraussetzung dafür ist zum Beispiel eine ausführliche Dokumentation der Entwicklung.

3.4 Vergleich von Software Asset Management Produkten

Auf dem Markt existieren eine Vielzahl von SAM-Produkten mit zum Teil unterschiedlichen Funktionen und Schwerpunkten. In diesem Abschnitt werden 4 Produkte näher betrachtet. Die Auswahl erfolgte im Hinblick auf einen möglichen Einsatz, da Produkte des Herstellers bereits bei der NÖ Landesverwaltung im Einsatz sind, andererseits wurde auch ein Produkt, das als Open Source zur Verfügung gestellt wird, ausgewählt. Aus der Gegenüberstellung der Anforderungen wird die Entscheidung abgeleitet ob die Implementierung eines der SAM-Produkte oder eine Eigenentwicklung für die NÖL sinnvoll ist.

3.4.1 Microsoft System Center Configuration Manager 2007

Der Microsoft System Center Configuration Manager (SCCM) ist die Verwaltungsplattform für IT-Umgebungen von Microsoft.
Der SCCM umfasst im Wesentlichen die folgenden Bereiche (vgl. [SCCM, 2010]):

- Betriebssystembereitstellung,
- Softwareverteilung,
- Softwareupdate,
- Software Asset Intelligence.

Im Rahmen dieser Betrachtung wird auf die Software Asset Intelligence Komponente eingegangen.
Das Programm erfüllt grundsätzlich die Anforderungen an ein SAM. SCCM beinhaltet Software und Hardware Inventarisierung. Anhand einer Wissensdatenbank werden die Daten analysiert und ggf. Software Suiten erkannt und als ein Produkt dargestellt. Die Wissensdatenbank umfasst beim SCCM ca. 400.000 Einträge. Die Kompatibilität beschränkt sich allerdings auf Windows Betriebssysteme. Weites kann auf Basis eines Softwareproduktes eine detaillierte Aufzeichnung der Softwarenutzung angestoßen werden. Wie bei Microsoft Programmen üblich ist das Berichtswesen sehr umfangreich. Allein in der Grundinstallation sind mehr als 300 Berichte enthalten. Die Berichte können über ein Webinterface angezeigt werden. Alle enthaltenen Berichte können frei angepasst und konfiguriert werden.

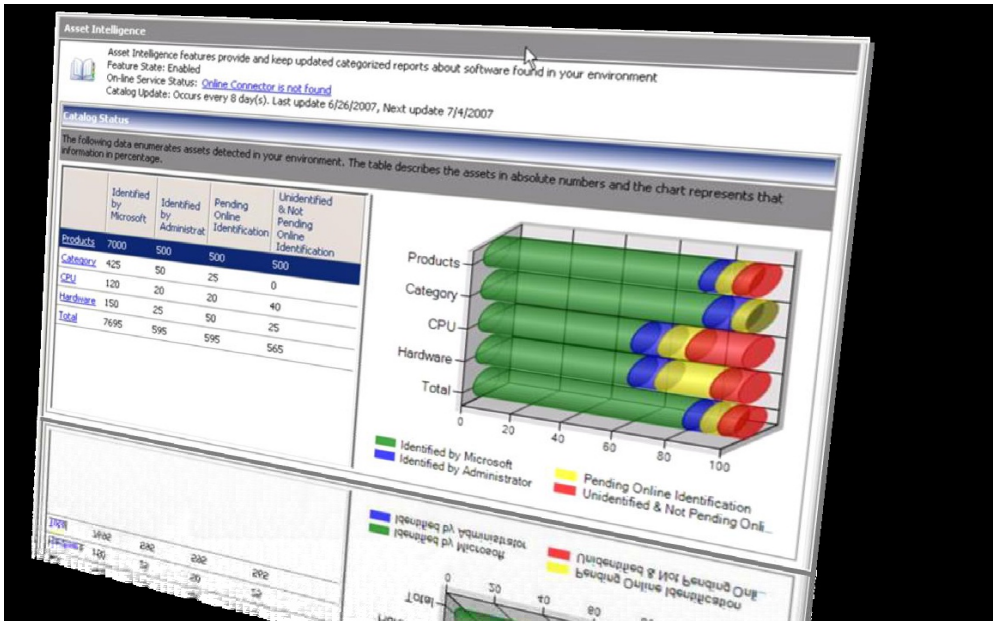


Abbildung 8: SCCM – Asset Intelligence[8]

Als Nachteil ist zu sehen, dass die Einbindung von bestehenden Daten im Unternehmen zum Soll-Ist-Vergleich nicht standardmäßig vorgesehen ist. Generell ist der SCCM nicht für die individuelle Erweiterung auf Kundenbasis ausgelegt. Auch die Unterstützung von Prozessen, zum Bespiel in Form eines Workflows, fehlt. Als Datenbank wird Microsoft SQL-Server ab Version 2005 unterstützt. Die Kosten für die NÖL durch die derzeitig gültigen Verträge als gering zu betrachten.

3.4.2 Novell ZENworks Asset Management

Die Software ist nach dem klassischen Aufbau eines SAM Produkts aufgebaut. Die Inventarisierung erfolgt durch ein Dienstprogramm, das als Service installiert wird. Das Dienstprogramm kann auch zur Erkennung eigener Produkte erweitert werden. Die Erkennung erfolgt ebenfalls anhand einer Wissensbank. Die Daten werden in einem Datenbanksystem abgespeichert. Es werden zurzeit Microsoft SQL Server und Oracle unterstützt. Des Weiteren umfasst die Software eine umfangreiche Anwendung zur Erfassung des Sollzustandes wie Lizenzen und Verträge. Diese können dann mit den Ist-Daten verglichen werden. Die Daten können über Schnittstellen importiert werden. Damit wird doppelter Eingabeaufwand vermieden, aber der Aufwand für den

[8] Bildquelle: [SCCM, 2010]

regelmäßigen Import der Daten bleibt. Hervorzuheben ist auch, dass alle gängigen Betriebssystemplattformen (Windows, Linux, MAC) unterstützt werden. Der Berichtsteil ist webbasierend, mit einigen Grafikoptionen (wie Torten oder Säulendarstellung) können die Berichte ansprechend dargestellt werden. Es stehen mehrere hundert Standardberichte zur Verfügung. Zu erwähnen sind die Softwarekonformitätsberichte (Abbildung 9) die die gefunden Softwareprodukte dem tatsächlichen Softwarekauf gegenüberstellen. Voraussetzung dafür ist aber, dass die Lizenzen in der Anwendung eingegeben oder importiert worden sind.

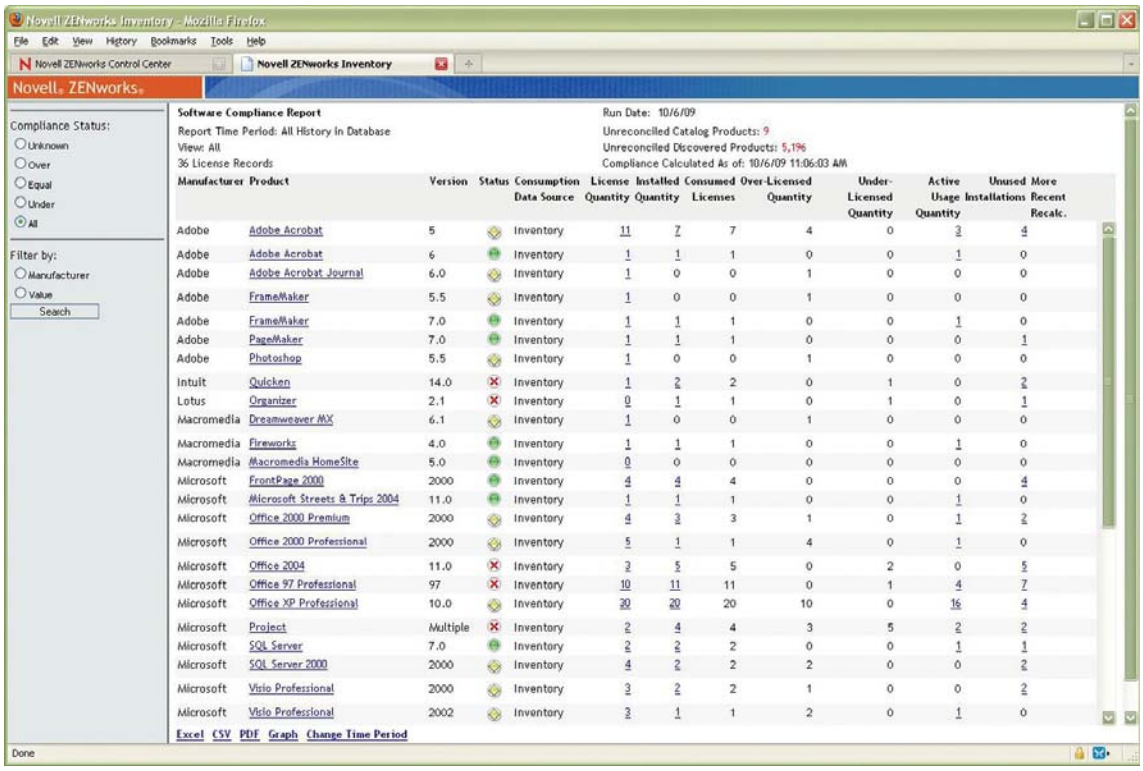


Abbildung 9: Softwarekonformitätsbericht[9]

Als Nachteil der Lösung sind die Kosten zu sehen, da das Lizenzmodell auf der Anzahl der inventarisierten Assets beruht.

3.4.3 Miss Marple 2010

Miss Marple 2010 ist eine kommerzielle Lösung des Softwarehersteller ADLON Datenverarbeitung GmbH und unterstützt einen ganzheitlichen Ansatz des Software Asset Management. Das bedeutet, dass sowohl die Pflege des Soll-Zustandes der Software

[9] Bildquelle: [ZENWorks, 2010]

und Hardware wie auch der Abgleich mit dem Ist-Zustand automatisiert unterstützt wird. Im Gegensatz zu den anderen Produkten im Vergleich unterstützt es auch die Prozesse des Software Asset Management wie sie in den internationalen Standards ISO 19700 und ITIL beschrieben sind. Die Prozesse werden durch einen im Produkt implementierten Workflow unterstützt. Als Zielplattformen sind die Betriebssysteme Windows, MAC-OS und diverse Linuxderivate unterstützt. Die Software bietet eine moderne und klar strukturierte Oberfläche wie in Abbildung 10 zu sehen ist.

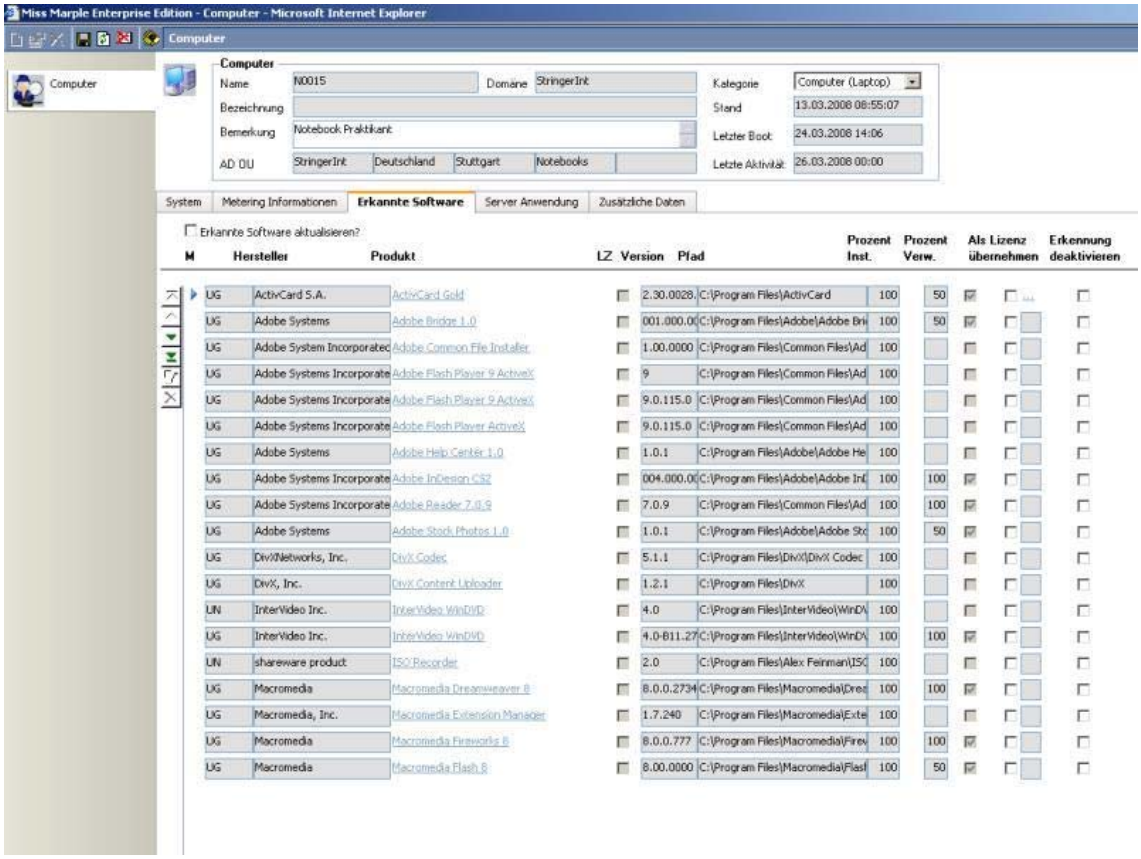


Abbildung 10: Softwareliste eines Computers[10]

Die Erkennung der installierten Softwareprodukte wird anhand einer Lizenzdatenbank durchgeführt. Die Lizenzdatenbank enthält circa 40.000 Softwareprodukte von 50 Softwareherstellern weltweit. Die Datenhaltung erfolgt in einer Microsoft SQL-Datenbank ab Version 2005.

[10] Bildquelle: [Marple, 2010]

Für das Berichtswesen sind die grundlegenden Reporte vorgefertigt. Weiters lassen sich Berichte über Reportgeneratoren wie die Microsoft Reporting Services oder Crystal Reports durchführen. Das Berichtswesen ist nicht so stark ausgeprägt wie bei den anderen beiden kommerziellen Produkten.

3.4.4 SpiceWorks

SpiceWorks ist der Open Source Vertreter bei den SAM-Lösungen. Das Hauptaugenmerk dieser Management-Software liegt in der Inventarisierung der Systeme und der Erkennung der darauf installierten Software, die sich im Netzwerk befinden. Durch dieses Programm hat der Anwender Zugriff auf Daten und Softwareversionen, die im Netzwerk installiert worden sind. Man kann allerdings auch sehen wie oft, wann und wo diese Daten heruntergeladen und installiert wurden. Der IT-Administrator ist dadurch in der Lage die IT-Umgebung zu verwalten und passende Aktualisierungen von Soft- oder Hardware auszuführen. Die Daten der überwachten Geräte sammelt SpiceWorks auf einem zentralen Management-Server. Dort werden sie in einer eigenen Datenbankstruktur abgelegt und verwaltet. SpiceWorks unterstützt nicht nur Computer mit den üblichen Betriebssystemen Windows, Mac OS X und Linux, sondern auch VoIP-Telefone, IP-Drucker und ähnliche Kleingeräte. Unter Windows kommt bei der Abfrage dieTechnik WMI[11] zum Einsatz. SpiceWorks bietet eine moderne und innovativ wirkende Benutzeroberfläche, die sich einfach bedienen lässt. Sie verwendet dynamische HTML-Techniken, und ähnelt dadurch dem Look and Feel der großen Webseiten.

[11] WMI: Schnittstelle zum Auslesen von Einstellungen bei Windows Betriebssystemen

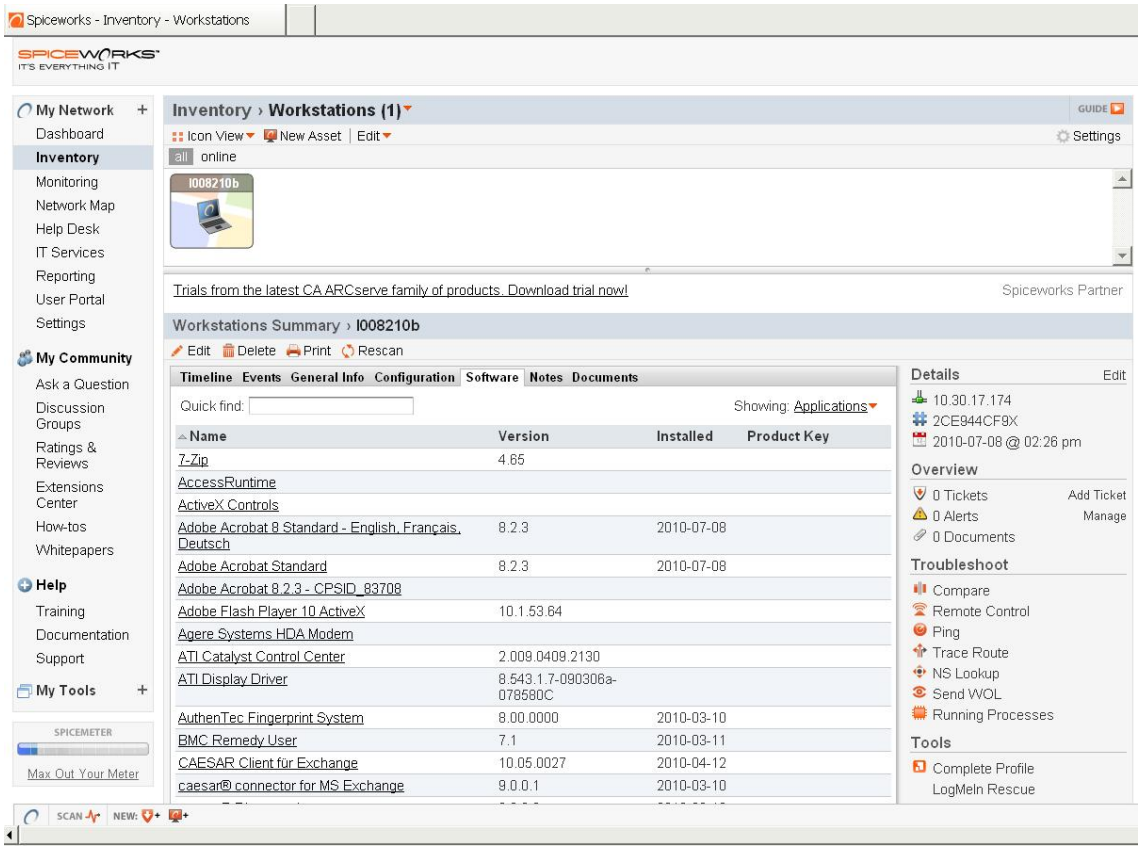


Abbildung 11: Moderne Benutzeroberfläche von Spiceworks

Als Nachteil von Spiceworks ist die Skalierbarkeit zu sehen. Der Hersteller selbst gibt an, dass die Lösung bis zu 1000 Geräten sehr gut ist und dann, auch bei entsprechenden Hardwareeinsatz, abnimmt. Weiters ist das Berichtswesen nicht ausgeprägt. Durch die eigene Datenbankstruktur ist auch das Erstellen von eigenen Berichten sehr aufwändig. Hier sind die kommerziellen Produkte klar im Vorteil.

3.4.5 Zusammenfassung

Zur besseren Übersicht werden die Systeme noch einmal in tabellarischer Form zusammengefasst und gegenübergestellt.

	SCCM	ZENWorks	Miss Marple 2010	Spiceworks
Grundfunktionalität	Sehr gut	Sehr gut	Sehr gut	Gut
Asset Management Prozess	Nein	Teilweise	Sehr gut	Nein
Logistische Prozesse	Nein	Teilweise	Teilweise	Nein
Verifikation und Einhaltung	Teilweise	Teilweise	Ja	Teilweise
Unterstützung von Workflow	Nein	Teilweise	Ja	Nein
Schnittstellen zur Einbindung von Daten	Nein	Importprogramme	Importprogramme AddIns	Nein
Skalierbarkeit	Sehr gut	Sehr gut	Sehr gut	Befriedigend
Kompatibilität (verfügbare Agents)	Nur Windows	Windows, Linux, MAC OS	Windows, Linux, MAC OS	Windows, Linux, MAC OS, IP-Geräte
Berichtswesen – im Produkt enthalten	Sehr gut	Sehr gut	Gut	Befriedigend
Lizenzierung	Je Client	Je Client	Paketpreis	OpenSource

Tabelle 1: Vergleich der Systeme

Die kommerziellen Produkte überzeugen durch eine gute Grundfunktionalität und modernen Benutzeroberflächen wobei der Schwerpunkt auf dem Bereich Verifikation liegt. Lediglich Miss Marple 2010 unterstützt durch eine eingebaute Workflow-Engine auch die Umsetzung von Prozessen. ZENWorks und Miss Marple 2010 unterstützen zusätzlich auch noch den Bereich Asset Management. Schwächen zeigen hingegen alle Produkte bei der Integration von bestehenden Daten. Es werden lediglich Schnittstellen zum Import von Daten zur Verfügung gestellt, sodass eine doppelte Datenhaltung nötig ist, um einen Abgleich der Soll-Ist-Situation durchzuführen. Die Vergabe von Berechtigungen ist bei allen Produkten möglich, aber doch sehr aufwändig wenn diese granular vergeben werden müssen.

Auch wird nicht durchgehend darauf geachtet, einfache Möglichkeiten zur Suche anzubieten, stattdessen muss oftmals über lange Listen ausgewählt werden, was bei tausenden von Clients die Benutzerfreundlichkeit deutlich einschränkt. Alle Produkte gehen von einer zentralen Administration aus. Schnittstellen für eine dezentrale Administration sind lediglich bei ZenWorks und Miss Marple in Form von Add-on-Programmierung vorgesehen.

3.5 Entscheidungsfindung: Eigenentwicklung oder Standardsoftware

Aus dem Vergleich der Standardlösungen für SAM-Produkte können zwei Produkte vorab ausgeschieden werden. Novells ZENWorks Asset Management ist auf Grund der Kosten nicht mehr weiter zu berücksichtigen, da die reinen Lizenzkosten bereits einen knapp sechsstelligen Betrag ausmachen würden. Das Open-Source-Produkt SpiceWorks ist auf Grund der mangelnden Skalierbarkeit auszuscheiden. Die Lösung sollte zehntausend Clients servicieren können, die Grenzen bei SpiceWorks liegen bei tausend Clients.

3.5.1 Allgemeines

Derzeit ist eine eindeutige Tendenz zu Standardsoftware in der Literatur zu erkennen. Begründet wird diese Tendenz unter anderem mit dem Investitionsschutz, der Zukunftssicherheit und der Flexibilität durch Customizing-Möglichkeiten. Meist werden auch die geringeren Kosten zu einer Eigenentwicklung hervorgehoben. Als weiterer Vorteil ist der schnelle Einsatz zu erwähnen, der durch eine Standardsoftware gegeben ist.
Die Zukunftssicherheit resultiert meist aus der Marktposition des Anbieters. Die Produkte werden technologisch als State-of-the-Art angesehen. Allerdings muss man jeden Versionswechsel mitmachen um auf Dauer davon profitieren zu können. Eine Vielzahl der Funktionen wird oft schlichtweg nicht genutzt beziehungsweise benötigt.

Das Anpassen der Standardsoftware an die Gegebenheiten des Unternehmens durch Customizing ist bei den meisten Produkten möglich, zieht aber hohe Projektkosten mit sich. Auch müssen diese angepassten Funktionen bei jedem Versionswechsel nachgezogen werden.

Im Gegensatz dazu muss bei einer Eigenentwicklung ein detailliertes Konzept ausgearbeitet werden. Das erfordert einen hohen Zeitaufwand bevor noch mit der eigentlichen Entwicklung begonnen werden kann. Dafür kann die Lösung optimal an die Be-

dürfnisse des Unternehmens angepasst werden, da nur jene Anforderungen implementiert werden die unbedingt nötig sind.

So kann man sagen, dass die Vorteile der einen Alternative die Nachteile der anderen sind und umgekehrt.

3.5.2 Gegenüberstellung der Lösungen - Entscheidung

Zur besseren Übersicht wurden die Anforderungen und die Erfüllung durch die Lösungen als Tabellezusammengefasst.

	SCCM	Miss Marple	Eigenent-wicklung
Grundfunktionalität	+	+	~
Schnittstelle zu RW-Assetmanagement	-	~	+
Verifikation und Einhaltung	+	+	+
Optimierte Abläufe	-	+	+
Dezentrale Administration und Rechte	-	~	+
Schnellere Einführung des Systems	+	~	-
Harwarekosten	~	-	+
Geringe Abhängigkeit von Fremdfirmen	-	-	+
Geringes Risiko für Unternehmen	~	+	~
Kosten	+	- (Projektkosten)	~

Tabelle 2: Bewertung der Alternativen

Auf Grund dieser Ergebnisse wird in der vorliegenden Arbeit die Eigenentwicklung eines Software Asset Managements als die bessere Lösung angesehen.

4 Systemkonzept

4.1 Allgemeines

Im Systemkonzept werden die Anwendungskomponenten des gesamten SAM-Systems entworfen und die eingesetzten Technologien erläutert. Im Zuge der Anforderungen an ein IT-System wird Wert auf die Verwendung von Standardsoftware und leichte Anpassbarkeit bzw. Erweiterung der Funktionalitäten wie Schnittstellen gelegt.

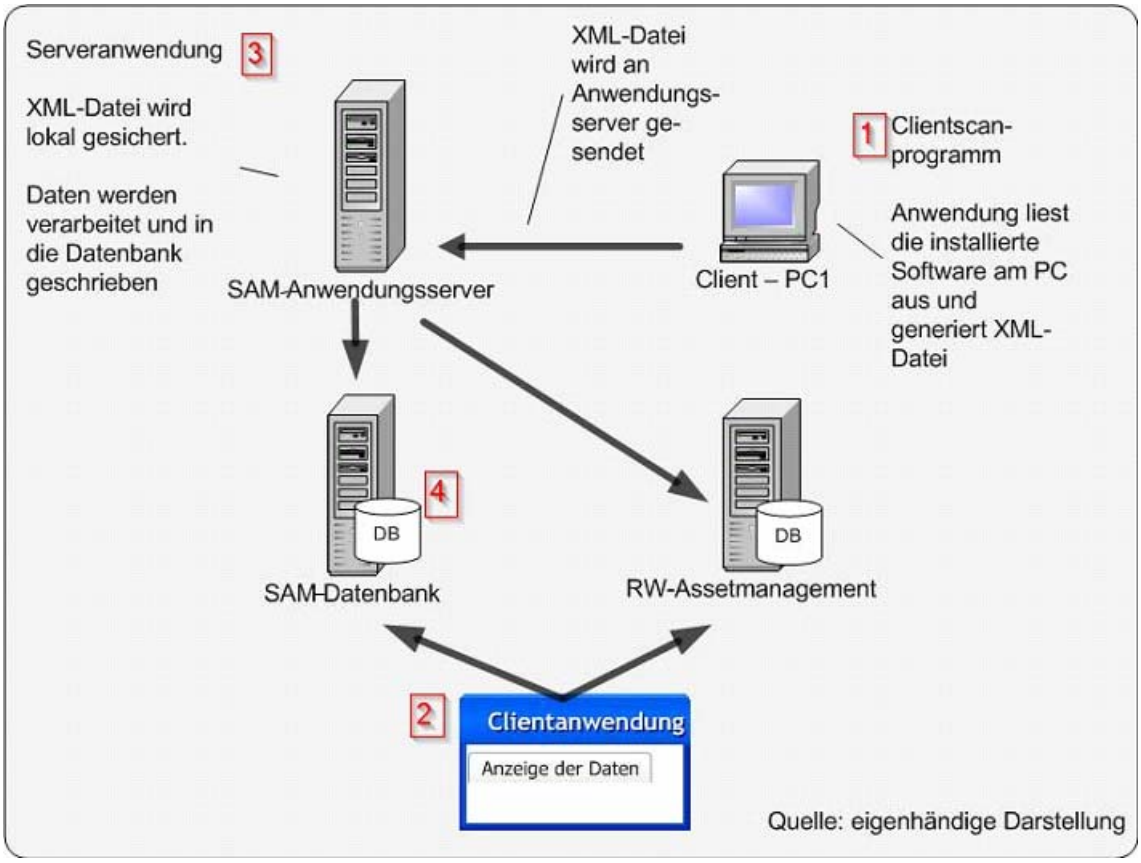


Abbildung 12: Systemkonzept

Wie in Abbildung 12 zu erkennen ist besteht das IT-System aus vier eigenständigen Komponenten:

1. Clientscanprogramm
   Das Scanprogramm liest die installierte Software am Client aus. Die Daten werden in einer XML[12]-Datei abgespeichert. Im Anschluss werden die Daten an die Serveranwendung zur Weiterverarbeitung gesendet.

2. Clientanwendung
   Die Clientanwendung dient zum Anzeigen der ermittelten Daten aus der SAM-Datenbank und der RW-Assetdatenbank. Hier ist besonders darauf zu achten, dass nur die für die jeweilig definierte Rolle berechtigten Daten angezeigt werden.

3. Serveranwendung
   Die Serveranwendung besteht aus zwei Komponenten. Die erste Komponente dient zum Abspeichern der Daten die von den Clients gesendet werden. Die zweite Komponente dient zur eigentlichen Verarbeitung der Daten und der Speicherung in der SAM-Datenbank und zum Bearbeiten der Stammdaten.

4. Datenbank
   Die anfallenden Daten für das IT-System werden in einer eigenen Datenbank abgespeichert.

4.2 Systemkonzept Clientscanprogram

4.2.1 Allgemeines

Das Clientscanprogramm ist je nach geforderter Betriebssystemplattform zu entwickeln bzw. abzustimmen. Primär werden für das SAM-System die Daten der Clients (PCs und Laptops) der NÖL benötigt. Die Betriebssystemlandschaft ist hier derzeit einheitlich Windows XP mit Service Pack 3. Um unterschiedlichste Plattformen, relativ rasch, mit wenig Aufwand zu unterstützen wurde bewusst ein sehr einfaches Design für das Scanprogramm gewählt. Das Grunddesign (siehe 4.2.3) kann daher generell, für die unterschiedlichen Plattformen, übernommen werden. Dies hat den Vorteil, dass je nach vorhandenen Know-how für die Implementierung beliebige Entwicklungsumgebungen

[12] XML: Extensible Markup Language

und Programmiersprachen wie etwa Visual Basic, C# oder Java verwendet werden können.

4.2.2 Grundlagen zur Ermittlung der installierten Software

4.2.2.1 Die Zukunft – Software Identification Tags

Eine der Kernaufgaben im Software Asset Management ist die eindeutige Identifizierung der Softwareprodukte auf den zu servisierenden Clients. Da die Informationsinhalte die ein Installationsprogramm bei der Installation abspeichert, nicht standardisiert sind ist es häufig nicht möglich eine exakte Angabe zu machen, welche Software in welcher Version wie oft installiert ist. Auch ist meistens eine tiefe Kenntnis des Produktes notwendig (siehe Kap. 3.3, Wissensdatenbanken von SAM-Produkten) um installierte Teilkomponenten dem richtigen Produkt zuzuordnen. Die Information benötigt man aber zur korrekten Lizenzverwaltung. Ende des Jahres 2009 hat die ISO den Standard ISO 19770-2 verabschiedet, der diesem Zustand ein Ende bereiten soll. Die Norm ist eine Erweiterung zur ISO 19770-1, die den Prozess des Software Asset Management (siehe Kap. 3.1), beschreibt. Kernstück ist die Definition eines Software Information Tag (SIT), eine XML-Datei, die an einer bestimmten Stelle hinterlegt werden muss. Software-Scanner können diese Informationen dann auslesen und an die jeweilige Inventarisierungslösung weiterleiten. Die SIT-Datei besteht aus zumindest sieben Pflichtfeldern:

1. entitlement_required_Indicator
   Gibt an ob eine kostenpflichtige Lizenz erforderlich ist
2. product_title
   Gibt den Produktnamen an
3. product_version
   Gibt die Version in der Form major, minor, build, review an
4. software_creator
   Hersteller der Software
5. software_licensor
   Lizenzgeber (z.B.: Tochtergesellschaft, Händler)
6. tag_creator
   Gibt an wer den SIT generiert hat (z.B.: Tochtergesellschaft, Händler)
7. software_id
   Der wichtigste Tag stellt eine eindeutige ID zur Verfügung (für Compliance-Prüfung, Abgleich mit anderen Managementsystemen)

Ein Beispiel einer SIT-Datei mit den Pflichtfeldern zeigt Abbildung 13.

```
<?xml version="1.0" encoding="UTF-8"?>
<swid:software_identification_tag
      xmlns:swid="http://standards.iso.org/iso/19770/-2/2009/schema.xsd"
      xmlns:xsi="http://www.w3.org/2001/XML-Schema-instance"
      xsi:schemalocation="http://standards.iso.org/iso/19770/
                          -2/2009/schema.xsd">
  <!-- Mandatory Elements -->
  <swid:entitlement_reqired_indicator>false</swid:entitlement_reqired_indicator>
  <swid:product_title>Firefox</swid:product_title>
  <swid:product_version>
    <swid:name>3.0.0.10</swid:name>
    <swid:numeric>
      <swid:major>3</swid:major>
      <swid:minor>0</swid:minor>
      <swid:build>0</swid:build>
      <swid:review>10</swid:review>
    </swid:numeric>
  </swid:product_version>
  <swid:software_creator>
    <swid:name>Mozilla Corporation</swid:name>
    <swid:regid>regid.1994-11.com.mozilla</swid:regid>
  </swid:software_creator>
  <swid:software_lizensor>
    <swid:name>Mozilla Corporation</swid:name>
    <swid:regid>regid.1994-11.com.mozilla</swid:regid>
  </swid:software_lizensor>
  <swid:software_id>
    <swid:unique_id>firefox.3.0.0.10</swid:unique_id>
    <swid:tag_creator_regid>regid.1994-11.com.mozilla</swid:tag_creator_regid>
  </swid:software_id>
  <swid:tag_creator>
    <swid:name>Mozilla Cooperation</swid:name>
    <swid:regid>regid.1994-11.com.mozilla</swid:regid>
  </swid:tag_creator>
</swid:software_identification_tag>
```

Abbildung 13: SIT-Datei von Firefox 3.0.0

Zusätzlich zu den Pflichtfeldern sind auch optionale Felder spezifiziert. Diese beschäftigen sich hauptsächlich mit der Softwarepacketierung und dem Lizenzmodell. Hier kann festgestellt werden ob eine Software im Rahmen eines Softwarepackets oder als Einzelanwendung installiert wurde. Ein weiterer Vorteil bildet sich auch aus der Tatsache das die SIT-Datei um für das eigene Unternehmen relevante Daten ergänzt werden kann. Dieser so genannte Tagging Prozess ist in Abbildung 14 dargestellt.

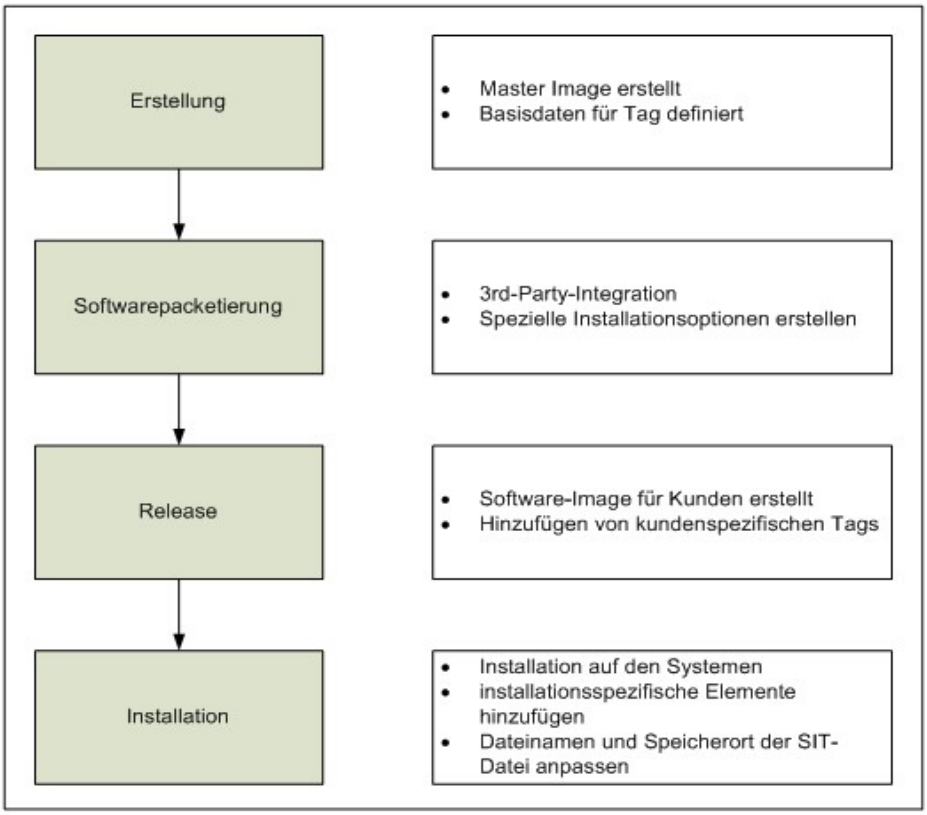


Abbildung 14: Tagging-Prozess

Da es sich hier um einen jungen Standard handelt ist seine Verbreitung noch nicht klar absehbar. Die großen Softwarehersteller wie Microsoft, Symantec und Adobe beginnen in ihre Softwarepakete die entsprechenden SIT-Dateien zu integrieren. Die Hersteller der Asset Management Suiten haben hier noch nicht nachgezogen. Positiv ist zu sehen, dass zum Auslesen der SIT-Dateien im Microsoftumfeld ab Windows Vista eine Schnittstelle zur Verfügung steht. Nach Einschätzung des Autors wird sich der Standard erst durchsetzen, wenn Open-Source-, und kleinere Hersteller diesen Standard in Ihre Produkte integrieren. Noch sieht die Realität anders aus.

4.2.2.2 Die Realität - Auslesen der installierten Software unter Windows XP

Unter Microsoft Betriebssystemen werden alle systemrelevanten Daten in der so genannten Registrierdatenbank abgespeichert. Die Registrierdatenbank unter Windows XP stellt die zentrale Konfigurationsdatenbank dar und enthält Information zur Verwaltung des Systems, sowie aller integrierter Systemdienste und –prozesse wie auch installierter Software und Parameter. Abbildung 15 zeigt die Hauptschlüssel der Registrierdatenbank.

Abbildung 15: Hauptschlüssel der Registrierdatenbank

Die Registrierdatenbank ist hierarchisch in Form einer Baumstruktur aufgebaut. Jeder Schlüssel kann wiederum Schlüssel oder konkrete Werte beinhalten. Abbildung 16 zeigt ein Beispiel um die Struktur zu verdeutlichen. Der Schlüssel HKEY_Current_USER\Software\NOEL enthält zwei weitere Schlüssel (NDS_ADS und Veränderung) und zwei Parameter (usrDC und usrKontext) mit gesetzten Werten.

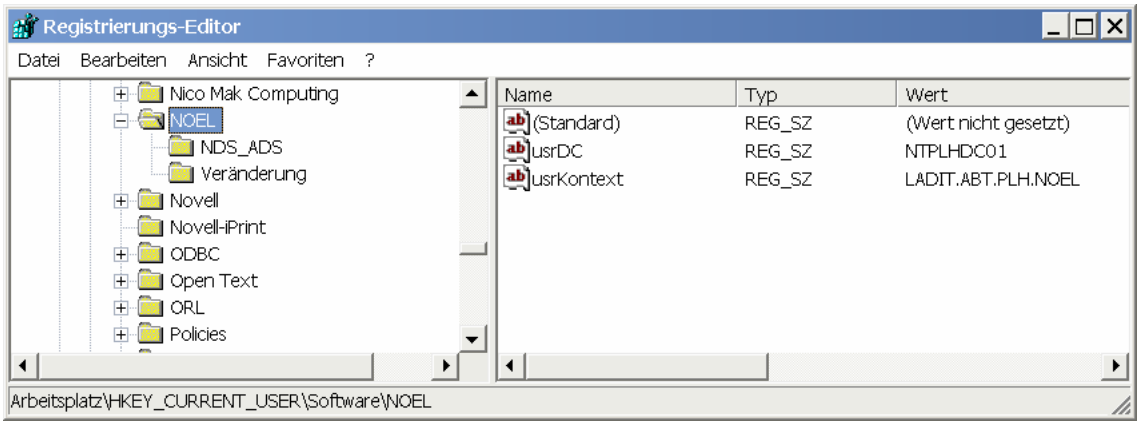


Abbildung 16: Beispiel Registrierdatenbank mit Schlüssel und Werten

Für den Zugriff auf die Registrierdatenbank sind ein Applikation Programming Interface (API) wie auch gekapselte Schnittstellen in den jeweiligen Frameworks zu den Programmierumgebungen vorhanden. Es stehen umfangreiche Funktionen zum Auslesen einzelner Schlüssel oder Werte wie auch zum Lesen aller Werte eines Schlüssels bereit. Weiters werden Funktionen zum Anlegen und Löschen von Schlüsseln und Werten zur Verfügung gestellt.

Zum Auslesen der installierten Software kann folgender Schlüssel verwendet werden:

HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall

Unter diesem Eintrag ist pro installierter Software ein weiter Schlüssel angelegt.
Dieser beinhaltet die Werte die das Setup-Programm abspeichert. Abbildung 17 zeigt einen Auszug der enthaltenen Informationen für das Softwareprodukt BMC Remedy:

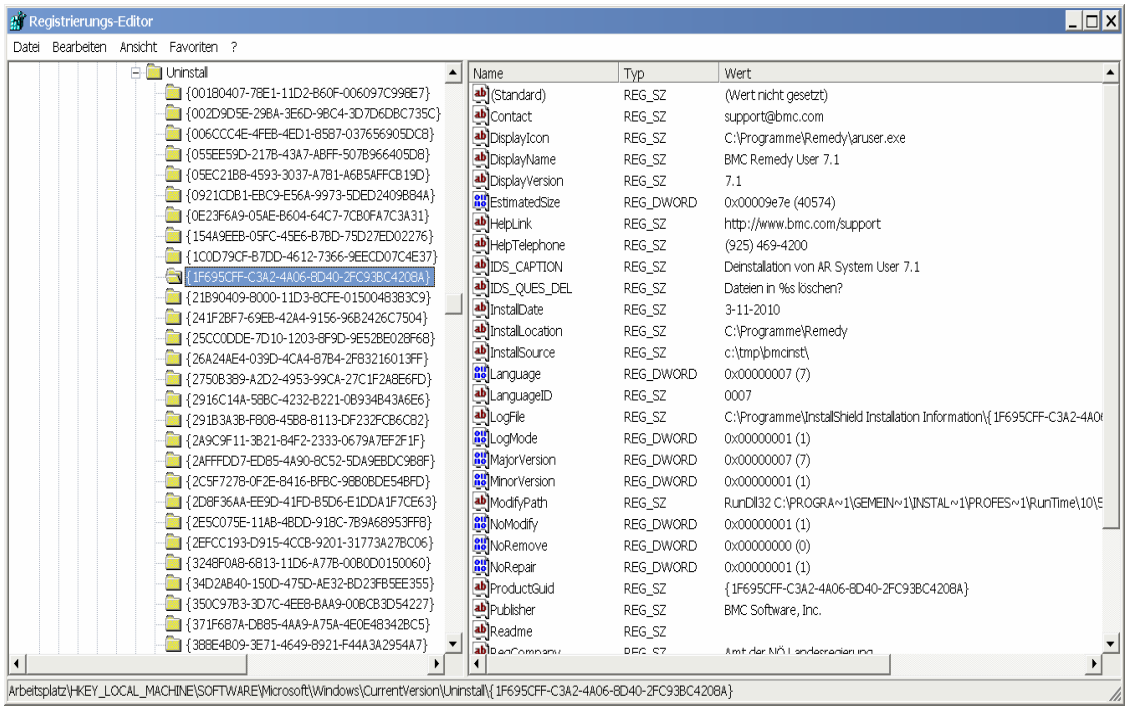


Abbildung 17: Auszug aus Registrierdatenbank

Da die Setup-Programme nicht einheitlich sind, muss man sich auf ein Minimum von Daten beschränken die generell verfügbar sind.
Für die Identifikation werden pro installierter Software folgende Werte ausgelesen:

- DisplayName,
- DisplayVersion,
- InstallDate,
- Publisher.

Unter dem Schlüssel HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows\-CurrentVersion\Uninstall werden aber nicht nur installierte Softwareprodukte sondern auch die installierten Patche für das Betriebssystem eingetragen. Diese Einträge sollen herausgefiltert und nicht übertragen werden.

4.2.3 Grunddesign

Bei Ausführung des Programms (im Standardfall bei Benutzeranmeldung) werden vorerst die erforderlichen Grunddaten des PCs ermittelt.

- Computername,
- IP-Adresse,
- MAC-Adresse,
- Benutzerkennung (Windowsanmeldung),
- Abteilung,
- Datum_Uhrzeit (Starzeitpunkt des Scans).

Diese Grunddaten stellen die minimalen Erfordernisse zur Identifizierung eines Assets dar. Mit der Ermittlung des Computernamens können die gescannten Clients im bestehenden RW-Assetmanagement identifiziert und zugeordnet werden. Es ist jederzeit möglich zusätzliche Informationen wie z.B.: Type oder Anzahl der CPUs, Aktualität der Virensignatur u.ä. auszulesen und zu übermitteln. Als Nächstes muss die installierte Software ermittelt werden. Die Daten werden unter Windows XP wie in Kap. 4.2.2 beschrieben aus der Windows Registrierdatenbank ausgelesen. Als Ergebnis wird eine XML-Datei erstellt. Der Aufbau der XML-Datei ist in Kap. 4.2.4 festgelegt. Danach erfolgt die Datenübertragung der XML-Datei zum Anwendungsserver. Durch diese Vorgangsweise ist die Art der Datenübertragung offen. So ist ein einfaches Kopieren oder Übertragen der Datei mittels ftp ebenso denkbar wie die Übertragung mittels Windows Sockets. Für die Implementierung unter Windows XP wurde eine Datenübertragung mittels Windows Sockets gewählt. Dies hat den Vorteil dass am Anwendungsserver keinerlei Zugriffsberechtigungen (z.B.: Dateisystem) für die Clients vergeben werden müssen. In Kap. 4.2.5 ist die Kommunikation mittels Windows Sockets beschrieben. Abbildung 18 zeigt den generellen Ablauf des Clientscanprogrammes. Für den Fall, dass ein Anwendungsserver nicht zur Verfügung steht wird die Möglichkeit implementiert, dass mehrere Anwendungsserver konfiguriert werden können. Das Clientscanprogramm versucht der Reihe nach die jeweiligen Anwendungsserver zu kontaktieren. An den Server, der als Erstes auf eine Verbindungsanforderung antwortet, werden die Daten übermittelt.

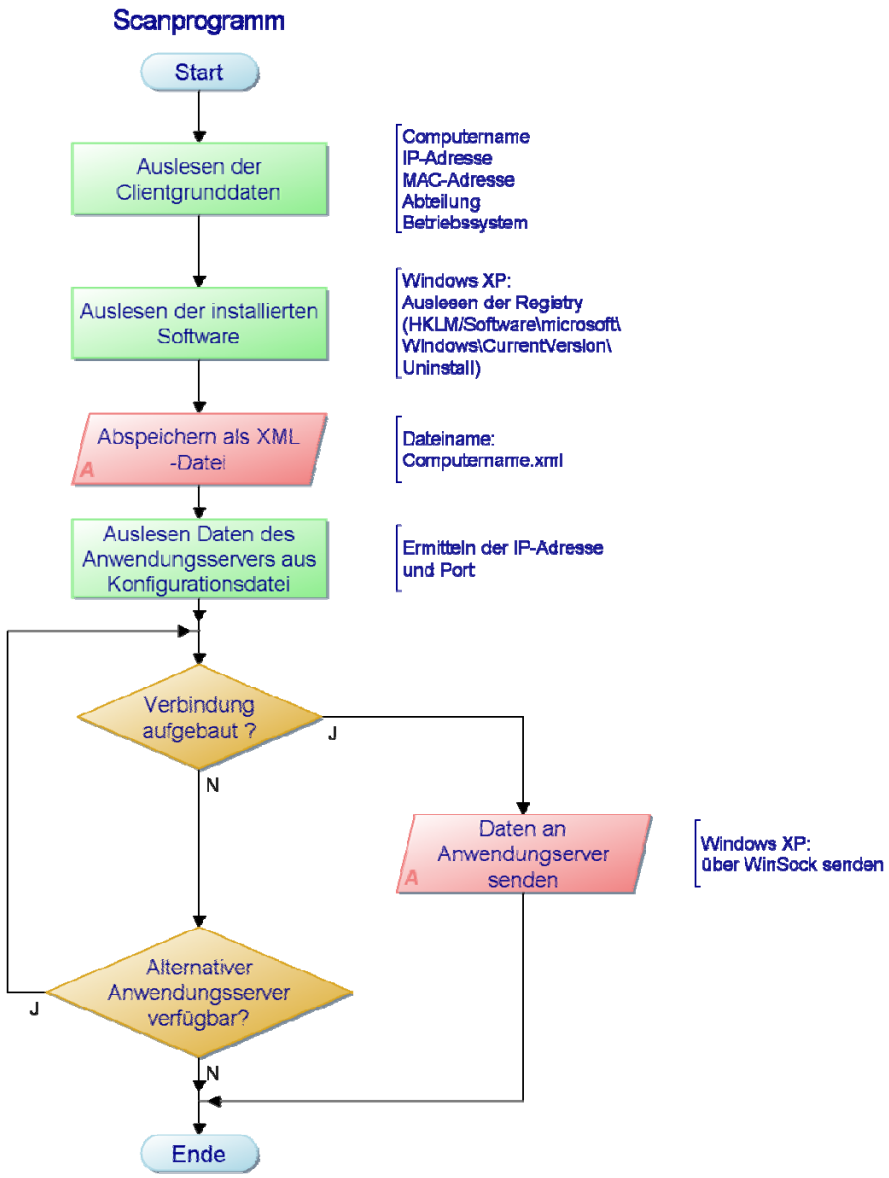


Abbildung 18: Flussdiagramm Scanprogramm

4.2.4 Struktur der XML-Datei

Wie im Systemkonzept beschrieben werden die gesammelten Daten in einer XML-Datei abgespeichert und an den Anwendungsserver übermittelt. Dies hat den Vorteil dass die Daten auf unterschiedlichsten Wegen auf dem Anwendungsserver abgelegt und verarbeitet werden können.

Das XML-Schema ist wie folgt aufgebaut:

```
<xsd:element name="PC-Grunddaten" type="xsd:string"/>
  <xsd:complexType mixed="true"/>
    <xsd:sequence>
      <xsd:element name="Computername" type="xsd:string"/>
      <xsd:element name="IP-Adresse" type="xsd:string"/>
      <xsd:element name="MAC-Adresse" type="xsd:string"/>
      <xsd:element name="Benutzerkennung" type="xsd:string"/>
      <xsd:element name="Abteilung" type="xsd:string"/>
      <xsd:element name="Datum_Uhrzeit" type="xsd:string"/>
    </xsd:sequence>


<xsd:complexType name="sw-Typ">
  <xsd:sequence>
    <xsd:element name="DisplayName" type="xsd:string"/>
    <xsd:element name="DisplayVersion" type="xsd:string"/>
    <xsd:element name="InstallDate" type="xsd:string"/>
    <xsd:element name="Publisher" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>


</xsd:element>

<xsd:element name="Software-liste">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="sw" type="sw-Typ" maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Die entsprechenden XML-Elemente würden dann wie folgt zusammengesetzt:

```
<PC-Grunddaten>
  <Computername>L008210B</Computername>
  <IP-Adresse>10.30.17.174</IP-Adresse>
  <MAC-Adresse>54-0C-47-2C-82-09</MAC-Adresse>
  <Benutzerkennung>syhw</Benutzerkennung>
  <Abteilung>LAD1-IT</Abteilung>
  <Datum_Uhrzeit>17.06.2010 08:30:21</Datum_Uhrzeit>
</PC-Grunddaten>

<software-liste>
  <sw>
    <DisplayName>BMC Remedy User 7.1</DisplayName>
    <DisplayVersion>7.1</DisplayVersion>
    <InstallDate>3-11-2010</InstallDate>
    <Publisher>BMC Software, Inc.</Publisher>
  </sw>
```

```
  <sw>
    <DisplayName>Symantec Endpoint Protection</DisplayName>
    <DisplayVersion>11.0.5002.333</DisplayVersion>
    <InstallDate>20100310</InstallDate>
    <Publisher>Symantec Corporation</Publisher>
  </sw>
  <!-- beliebige viele Software-Elemente -->
</software-liste>
```

4.2.5 Dateiübertragung mittels Winsock

Dieses Kapitel beschreibt die Kommunikationsmöglichkeit über Windows Sockets (WinSock) zwischen Clients und Server.

4.2.5.1 Definition Client und Server

Zur Übermittlung von Informationen über einen Kommunikationskanal sind zwei Stellen erforderlich. Zum Einem der Sender der den Kommunikationskanal öffnet indem er eine Gegenstelle anruft und zum Anderen der Empfänger der diesen Anruf entgegennimmt und antwortet. Ähnlich wie bei einem Telefongespräch differenziert man auch in der Netzwerkkommunikation und unterteilt die an der Kommunikation beteiligten Stellen, nämlich in den *Client* und den *Server*. Der Client initiiert die Kommunikation während der Server auf Anfragen wartet und dann den Clients antwortet. Entscheidend hierbei ist, dass der Client die Adresse des Servers kennen muss, nicht aber umgekehrt. Ob ein Programm auf einem Rechner als Client oder als Server agiert entscheidet darüber, wie das Programm die Socket API verwendet, um mit dem so genannten Peer zu kommunizieren. Für einen Server ist der Peer der Client und umgekehrt.

Neben der IP-Adresse des Servers, die der Client unbedingt kennen muss, fällt dem Port ebenfalls eine wichtige Rolle für eine geregelte Kommunikation über TCP zu. Die IANA[13] weist deshalb bestimmten Diensten jeweils fest definierte Ports zu. So sind gängige Ports, wie beispielsweise Port 21 (FTP) und 80 (HTTP) bereits reserviert. Dennoch können sich Client und Server entsprechend arrangieren, so dass prinzipiell jede Port-Nummer verwendet werden kann. Dies setzt allerdings voraus, dass der Client die Port-Nummer einer bestimmten Anwendung zuordnen kann.

[13] Internet Assigned Numbers Authority

4.2.5.2 Begriff Socket

Nach [WIKI03, 2010] ist ein Socket wie folgt definiert:

> „Ein Socket (abgel. von engl. Sockel oder Steckverbindung) ist ein Software-Modul mit dessen Hilfe sich ein Computerprogramm mit einem Rechnernetz verbinden und dort mit anderen Prozessen Daten austauschen kann. Die Kommunikation über Sockets erfolgt in der Regel bidirektional, d.h. über das Socket können Daten sowohl empfangen als auch gesendet werden. Sockets werden auch verwendet, um zwischen Prozessen auf demselben Computer Daten auszutauschen (Interprozesskommunikation)."

Davon ausgehend kann man sich einen Socket als eine "Steckdose" vorstellen. Rechner können über diese Steckdose mit anderen Rechnern kommunizieren. So gesehen ist ein Socket eine Abstraktion die es einer Anwendung ermöglicht sich in ein bestehendes Netzwerk einzuklinken um mit einer anderen Applikation zu kommunizieren. Für den Zugriff auf ein Socket stehen im Allgemeinen standardisierte Funktionen seitens des Betriebssystems zur Verfügung. Die erste Socket Implementierung gab es unter dem Betriebssystem FreeBSD und wurde unter dem Namen Berkeley Sockets API bekannt.

Man kann zwei Arten von Sockets unterscheiden:

- Stream Sockets (basieren auf dem Transmission Control Protocol (TCP))
- Datagram Sockets (basieren auf dem User Datagramm Protocol (UDP))

4.2.5.3 Ablauf der Kommunikation

Der Beginn der Kommunikation geht immer vom Client aus. Der Client erzeugt einen neuen Socket. Danach versucht er zum Server (IP-Adresse, Port) eine Verbindung aufzubauen. Sobald der Server die Verbindung akzeptiert ist die Verbindung aufgebaut und es können Daten ausgetauscht werden. Nach Abschluss der Verarbeitung wird der Socket wieder geschlossen. Abbildung 19 zeigt den prinzipiellen Ablauf der Kommunikation.

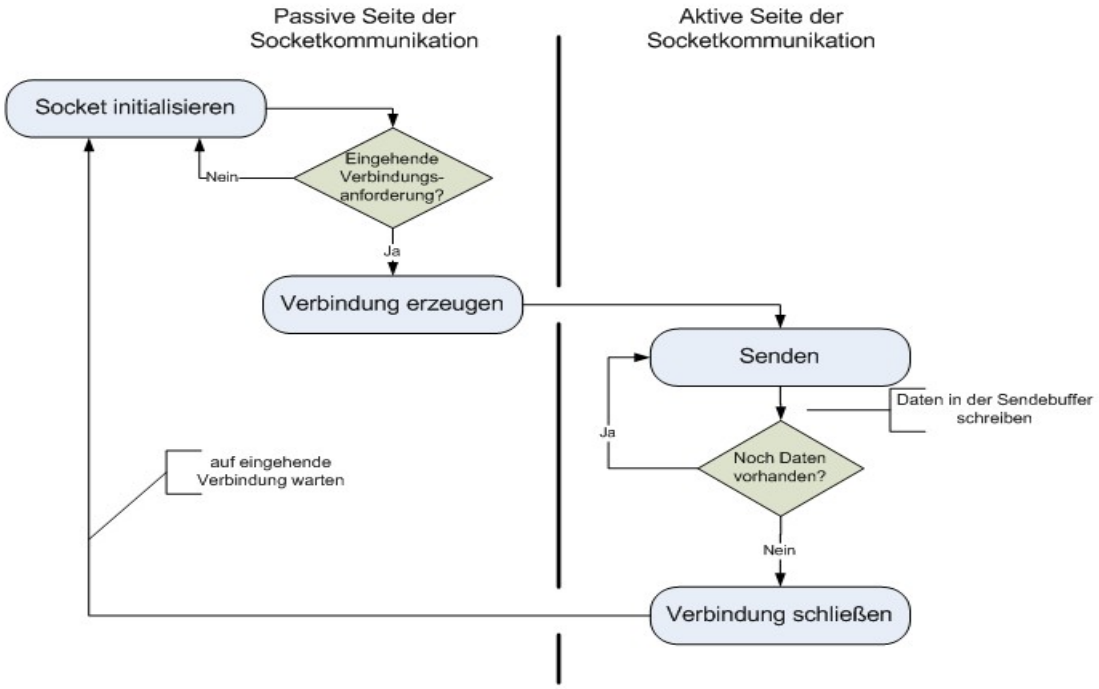


Abbildung 19: Client/Serverkommunikation über Sockets

4.3 Systemkonzept Clientanwendung

4.3.1 Allgemeines

Die Clientanwendung dient zur Darstellung der erfassten Daten und fungiert als Schnittstelle für die Benutzer. Je nach Rolle des aufrufenden Benutzers werden die Daten angezeigt für die er berechtigt ist. Wie in den Anforderungen in Kap. 3.4.2 beschrieben soll für das Benutzerinterface die WPF verwendet werden. Dabei ist näher auf die Konzepte der WPF einzugehen. Da die Beschreibung der WPF als Ganzes im Rahmen dieser Arbeit nicht möglich ist wird auf die Grundkonzepte eingegangen und die Eignung von WPF für datenorientierte Anwendungen diskutiert.

4.3.2 Was ist WPF

Mit dem Namen Windows Presentation Foundation zeigt Microsoft schon die Richtung wo sie den Haupteinsatz sieht, nämlich in der Präsentation bzw. grafischen Darstellung von Informationen zum Benutzer. Bei der Entwicklung handelt es sich aber nicht um eine Weiterentwicklung der bestehenden Möglichkeiten sondern um eine komplette Neuentwicklung.

Zwei Ansätze erscheinen auf den ersten Blick als wesentlich warum Entwickler auf die neuen Möglichkeiten umsteigen sollten. Als Erstes die Möglichkeit der Trennung von Design und Programmlogik und als Zweites der Umstieg von Pixel-Basierender Grafik auf Vektor-Basierende Grafik.

4.3.2.1 Trennung von Darstellung und Programmcode

Bisher gab es immer eine enge Verzahnung zwischen der Darstellung und der Programmlogik (Code-Behind). Man musste mit der jeweiligen Entwicklungssprache (z.B.: C#, VB.Net) das Aussehen der Darstellung ändern. Das bedeutet, in gewissem Ausmaß, dass der Grafiker programmiert und der Programmierer die Grafiken erstellt. Mit XAML (siehe 4.3.2.3) steht jetzt ein einheitliches Dateiformat für Grafiker und Entwickler zur Verfügung was den Austausch und die Zusammenarbeit wesentlich erleichtert. Jeder kann seine eigenen Tools verwenden. Die Entwicklung der Programmlogik und der Benutzeroberfläche einer Anwendung kann nun getrennt mit verschiedenen Werkzeugen erfolgen (Abbildung 20). Hier ist auch zu erkennen, dass für das Design nicht unbedingt Visual Studio nötig ist sondern auch andere Produkte in Frage kommen. Zu erwähnen wäre hier Microsofts Expression Blend das einen ausgereiften Designer für WPF darstellt und sich ähnlich wie Flash[14] bedienen lässt.

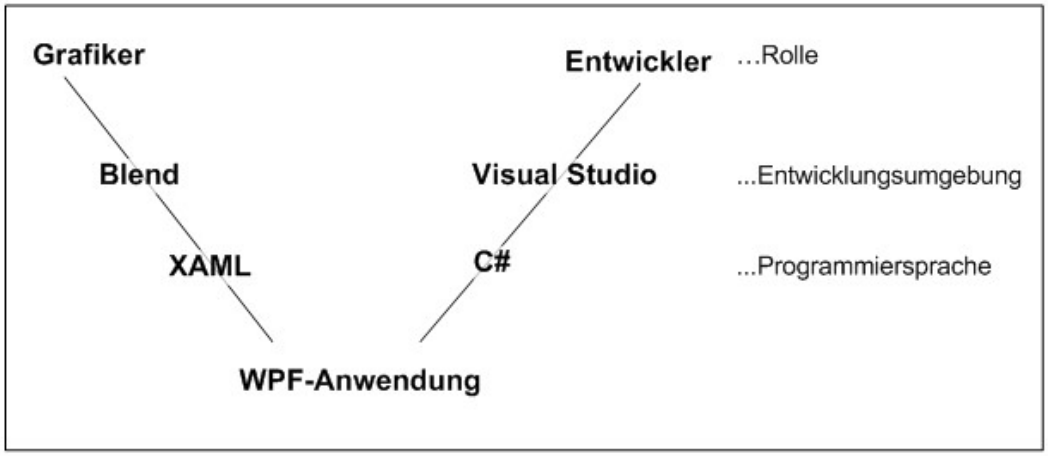


Abbildung 20: Getrennte Vorgehen bei Entwicklung und Design

Um die Verflechtung von Darstellung und Programmierung so gering wie möglich zu halten wurden verschiedene Techniken in XAML integriert. Abbildung 21 zeigt die Techniken die benutzt werden. Eine Technik, nämlich die der Datenbindung, ist für

[14] proprietärer Standard zur Entwicklung multimedialer Inhalte von Adobe

datenorientierte Anwendungen besonders interessant und wird in Kap. 4.3.3 näher betrachtet.

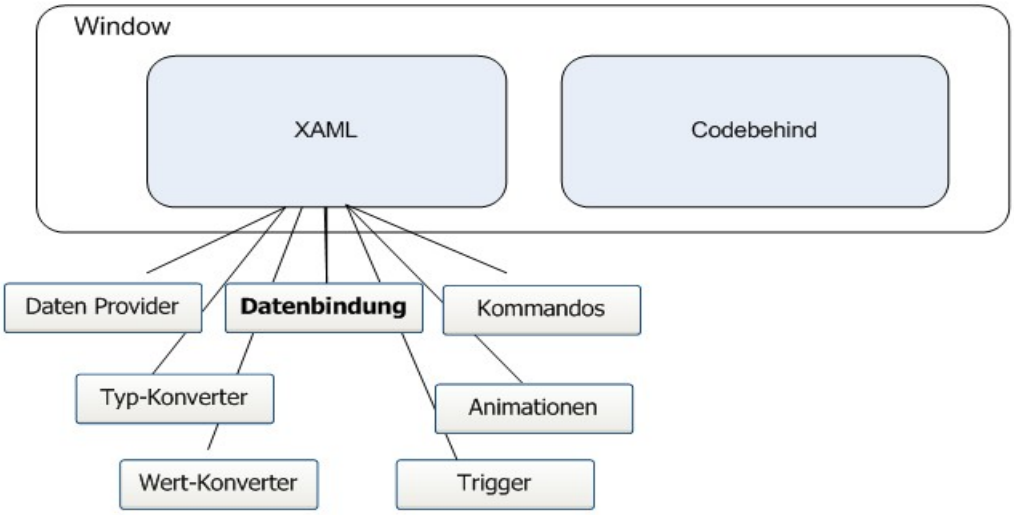


Abbildung 21: Trennung von Design und Programmierung

4.3.2.2 Vektorgrafiken anstatt Pixeldarstellung

Eine der augenscheinlichsten Unterschiede zu den bisherigen Möglichkeiten der Darstellung ist aber der Umstieg von Pixelorientierter Darstellung zu vektorbasierter Grafik. Bis jetzt wurden Anwendungen für gewisse Auflösungen optimiert, z.B.: 1280 x 800. Je nach sichtbarer Diagonale (z.B.: 17 Zoll oder 20 Zoll) wird die Anwendung größer oder kleiner dargestellt. Noch deutlicher werden die Unterschiede bei den Auflösungen der heutigen Notebookmonitore die eine Auflösung von 1680 x 1050 Bildpunkten auf 14-Zoll Monitor darstellen. Ein weiteres Problem ist, dass die Hersteller ihren Monitor auf eine Auflösung optimieren, andere Auflösungen werden hingegen interpoliert. Dies führt zu einer unscharfen Darstellung. Mit dem Übergang zu Vektorgrafiken kann sich jetzt die Anwendung oder das Betriebssystem um die Auflösung der Anwendung kümmern, der Monitor selbst kann in der für ihn optimierten Auflösung arbeiten. Als Entwickler selbst muss man kaum mehr Aufmerksamkeit darauf legen, da die WPF von Grund auf Vektorgrafiken basiert. Für die Größenänderung ist nicht der Entwickler zuständig sondern die WPF erledigt dies automatisch.

4.3.2.3 XAML - Extensible Application Markup Language

Mit der Einführung von WPF wurde auch die neue deklarative Sprache XAML für die Definitionen der Benutzeroberflächen eingeführt. XAML basiert auf dem offenen Stan-

dard XML. Die Benutzeroberfläche wird jetzt nicht mehr direkt in der jeweiligen Entwicklungsumgebung sondern in XAML definiert. Ein Designerprogramm braucht also nur die XAML-Dateien auszulesen und nicht den jeweiligen Programmdialekt. XAML wurde zwar mit WPF eingeführt und eng miteinander verbunden, kann aber auch in anderen Anwendungen verwendet werden.

Das XAML so vielseitig einsetzbar ist liegt auch daran, dass in XAML nur die Klassen selbst beschrieben werden, aber nicht die Oberfläche. Ein Beispiel soll dies verdeutlichen. Hier wird die Klasse *Button* instanziiert und die Eigenschaft *Content* auf „Hello World“ gesetzt.

```
<Button Content="Hello World" />
```

Obige Zeile ist die XAML-Deklaration eines Buttons mit der Beschriftung „Hello World“. Dazu alternativ der Ansatz in programmatischer Technik:

```
Button btnButton = new Button();
btnButton.Content = „Hello World“;
```

Da XAML auf XML basiert, ist die Struktur streng hierarchisch aufgebaut. Jeder Knoten hat einen übergeordneten und beliebig viele untergeordnete Knoten. Diese Struktur wird im XAML auch als Logical Tree bezeichnet. Der Logical Tree beinhaltet die Elemente bis auf Steuerelementeniveau. Abbildung 22 zeigt die Baumstruktur einer XAML-Definition.

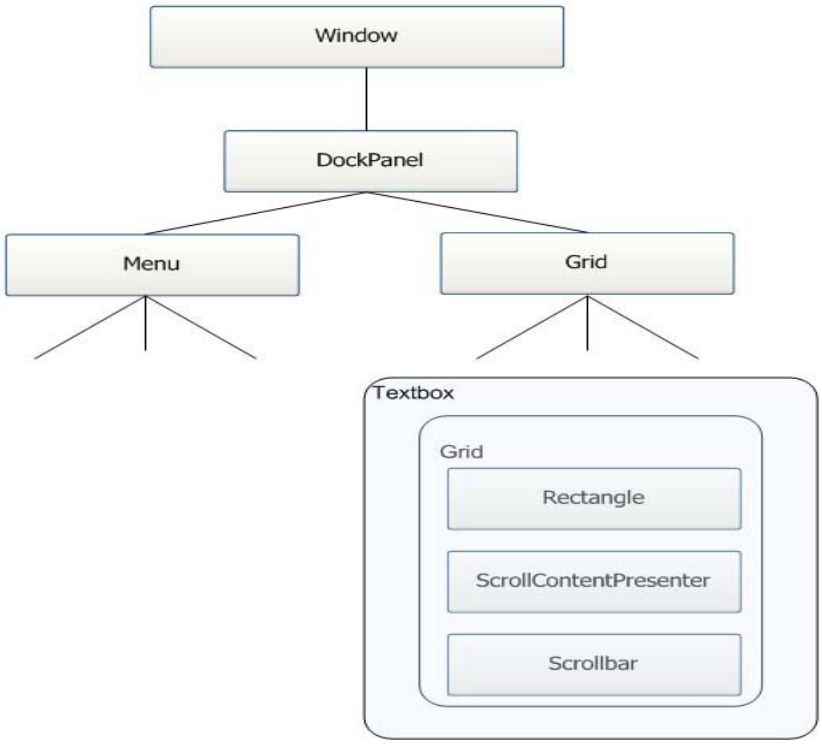


Abbildung 22: Baumstruktur XAML-Definition

Da ein Steuerelement aber wieder aus anderen Elementen (Grid, Rectangle, …) bestehen kann gibt es noch den Visual Tree der auch diese Elemente berücksichtigt. Der Visual Tree ist damit weit komplexer und feiner verästelt als der Logical Tree. Mit Programmcode kann aber nur auf den Logical Tree zugegriffen werden. Natürlich gibt es auch hier wieder einen Ausnahme (VisualTreeHelper) deren Verwendung aber nicht empfohlen ist.

Zu den Grundlagen beim Arbeiten mit XAML zählt auch das Einstellen der Eigenschaften der angesprochen Klassen, sprich Steuerelemente. XAML ist hier generell case-sensitiv und unterscheidet daher Groß-/Kleinschreibung bei den Eigenschaftsnamen. Am Beispiel des Setzens von Eigenschaften eines Buttons soll die Syntax verdeutlicht werden.

```
<Button Name="btnButton"
   Width="120" Height="40"
   Margin="34,61,24,61">
   Content="Hello World" />
```

Hier werden die Eigenschaften als Attribute gesetzt. Alternativ gibt es noch die Darstellung als Property Element die bei komplexeren Definitionen Verwendung findet.

```
<Button Name="btnButton">
   <Button.Width="120"</Button.Width>
   <Button.Content="Hello World"</Button.Content>
</Button>
```

4.3.2.4 Ereignisse (Routed Events)

Durch die umfangreiche Änderung die die WPF bei der Implementierung von Steuerelementen mit sich bringt musste auch die Behandlung von Ereignissen geändert werden. Diese nennt man jetzt Routed Events (vgl. [Mosers, 2010]). Bei Windows Forms Applikationen war es eindeutig welches Klickereignis auszulösen ist, nämlich jenes Steuerelement auf das der Mauszeiger zeigte. In der WPF kann jetzt aber ein Steuerelement verschachtelt sein. Eine Schaltfläche kann beispielsweise aus einem Text und/oder einer Grafik bestehen. Das Ereignis muss daher entsprechend auf das richtige Steuerelement gerouted werden. Das Routing passiert wiederum auf dem Visual Tree. Abbildung 23 zeigt wie ein Ereignis durch den Visual Tree durchgereicht wird.

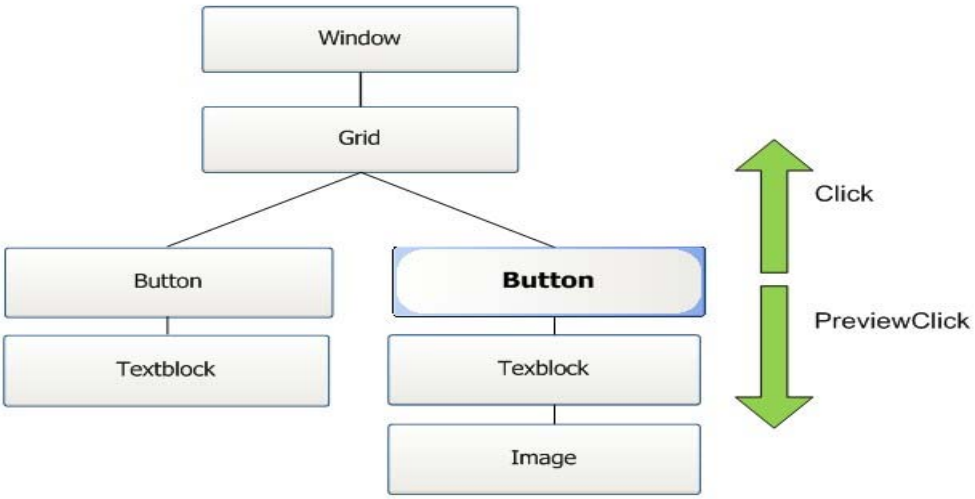


Abbildung 23: Darstellung des Routings bei Ereignissen

In den meisten Fällen muss der Programmierer keine Kenntnis darüber haben, aber es kann in besonderen Fällen durchaus sinnvoll sein auch in den verschachtelten Steuerelementen auf ein Ereignis zu reagieren.

4.3.2.5 Eigenschaften - Dependency Properties

Auch bei den Eigenschaften mussten durch die neuen Konzepte in der WPF Änderungen durchgeführt werden. Glücklicherweise ist es auch hier so, dass es aus Sicht der Programmlogik keine Änderung gibt. Der gewaltige Unterschied liegt aber im Detail. Herkömmliche Eigenschaften werden in privaten Variablen in ihren Klassen gespeichert. Wenn die WPF mit diesen Eigenschaften arbeiten muss, müsste über den Mechanismen der Reflexion[15] auf diese zugegriffen werden. Um in XAML direkt oder auch bei Datenbindung auf Eigenschaften optimal zugreifen zu können, wurden die Dependency Properties eingeführt. Bei den Dependency Properties werden im Unterschied zu den herkömmlichen Eigenschaften die Werte nicht mehr in den privaten Variablen der Klasse gespeichert, sondern alle Werte in einem zentralen Speicher abgelegt. Der zentrale Speicher ist über die Basisklasse *Dependceny-Object* abgebildet. Darum ist es auch eine Voraussetzung, wenn eine Klasse Dependency Properties einsetzen will, dass sie von der Basisklasse *Dependceny-Object* abgeleitet ist. Die Dependency Properties stellen folgende zusätzliche Funktionen zur Verfügung:

- eine automatische Aktualisierung,
- eine integrierte Validierung,
- die Deklaration von Standardwerten,
- den Aufruf von Callback-Methoden, wenn Wertänderungen aufgetreten sind.

In XAML wie auch bei Datenbindung kann auf einfache Art und Weise mit einem Dependency Property gearbeitet werden.
Im Visual Studio erkennt man ein Dependeny Property, dass es statisch definiert und in der Beschreibung immer ein Hinweis darauf vermerkt ist (siehe Abbildung 24).

[15] Reflexion: Eine Möglichkeit mittels Metadaten mit Eigenschaften, Events von Objekten zu arbeiten

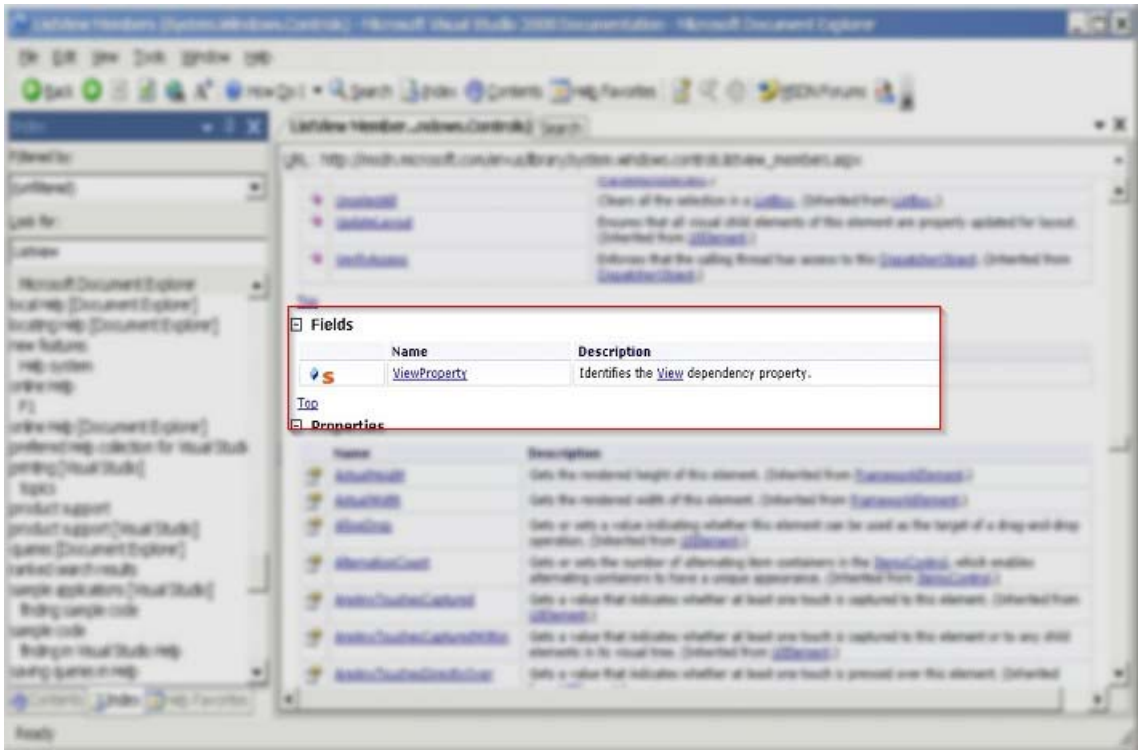


Abbildung 24: Dokumentation von Dependency Property im Visual Studio

4.3.3 WPF und Datenbindung

Wenn man die Steuerelemente durchsieht die im Visual Studio zur Verfügung stehen, wenn ein WPF-Projekt erstellt wird, stellt man fest, dass es keine datenorientierte Steuerelemente wie bei einem Windows-Forms Projekt gibt. Dies mag abschrecken, da man Steuerelemente wie das Datagrid gewohnt war. Die WPF hat aber grundlegende Mechanismen zur Datenbindung in Ihrem Fundament verankert. Dritthersteller, wie Xceed aber auch Microsoft selbst, haben zwar drauf reagiert und stellen ein Datagrid zur Verfügung. Bei der Entwicklung von WPF-Anwendungen sollte man sich aber doch mit den neuen grundlegenden Konzepten zur Datenbindung beschäftigen, da dies großes Potential und Vereinfachungen bringen kann und das Konzept der Trennung von Oberfläche und Programm unterstützt.

4.3.3.1 Der Datenkontext

Bei der Darstellung beziehungsweise Präsentation von Daten müssen diese dem Steuerelement zugewiesen werden. Dies geschah bisher dadurch, dass der jeweiligen Eigenschaft ein entsprechender Wert zugewiesen wurde.

```
txtComputerName = computer.Name;
txtComputerIP = computer.IpAdresse;
```

Dieses Beispiel soll zeigen, wie eng Programm und Anzeige miteinander gekoppelt ist. In XAML gibt es dafür, als neues Konzept, die Datenbindung über den Datenkontext. Der Datenkontext stellt einen Container dar, der jegliche Daten aufnehmen kann. Der Container kann über die Eigenschaft *DataContext* angesprochen werden. Die Eigenschaft ist in jedem Steuerelement vorhanden. *DataContext* ist vom Typ *object* sodass jegliche Daten zugewiesen werden können.

```
ComputerTextBox.DataContext = computer;
```

Es wurde dem *DataContext* das Objekt *computer* zugewiesen. Die Daten stehen dem Steuerelement zur Verfügung. Das bedeutet aber noch nicht dass sie angezeigt werden. Dazu stehen weitere Möglichkeiten zur Verfügung.

Eine Besonderheit ist, dass der Datenkontext, wenn er eingestellt wurde, an alle untergeordneten Steuerelemente weitergereicht wird. Dies geschieht wiederum mit dem Visual Tree der die Daten nach unten reicht. Das Weiterreichen wir durch Setzen eines Datenkontextes in einem unterordneten Steuerelement unterbrochen. Somit könnte man der Klasse *Window* den Datenkontext zuweisen und alle unterordneten Steuerelemente hätten Zugriff auf die Daten. Abbildung 25 zeigt dieses Prinzip. Die Daten die dem Datenkontext des *Window* zugeordnet sind stehen auch den Steuerelementen Grid, Button, Stackpanel, Checkbox und TextBox zur Verfügung. Durch geschickte Wahl des Datenkontextes können somit alle Steuerelemente mit den Daten die sie anzeigen sollen versorgt werden.

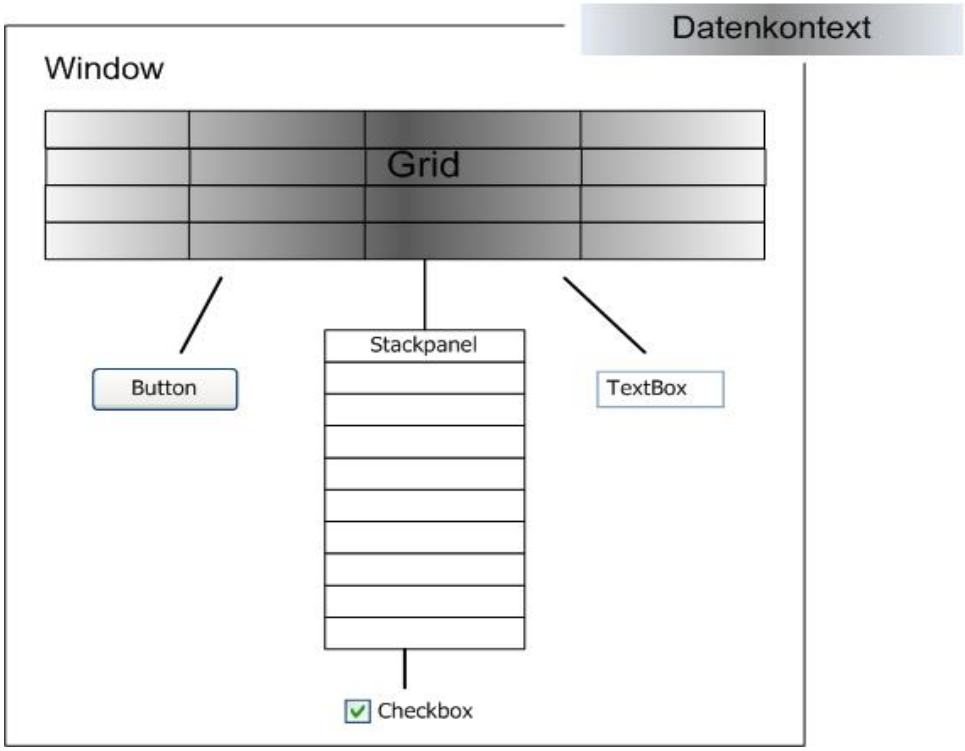


Abbildung 25: Weiterreichen des Datenkontextes über den Visual Tree

4.3.3.2 Die Datenbindung

Wie schon erwähnt wurden die Daten über den Datenkontext nur zur Verfügung gestellt. Für die Präsentation stellt die WPF die so genannten Datenbindungen zur Verfügung. Mit einer Datenbindung wird eine Verbindung zwischen der Datenquelle und dem Steuerelement definiert welche die Daten anzeigen soll. Es wird somit definiert wo die Daten die durch den Datenkontext bereitgestellt werden angezeigt werden. Hier zeigt sich auch der Vorteil, da jetzt alle Eigenschaften eines Steuerelementes mit der Datenquelle verbunden werden können. Einzige Voraussetzung ist, dass die Eigenschaft als Dependency Property definiert ist. Durch diese Erweiterung ist es sehr leicht möglich, auch Eigenschaften die das Aussehen eines Steuerelementes betreffen über Datenbindungen zu verändern. Nachfolgende Beispiele sollen die Möglichkeiten verdeutlichen:

```
<TextBox Computername="{Binding Computername}" />
<Button IsEnabled="{Binding IsEnabled}" />
<Rectangle Width="{Binding Widthobject}" />
```

Bis jetzt wurde die Datenbindung aber immer nur im Kontext des Anzeigens von Daten benutzt, auf Eingaben des Anwenders wurde noch nicht reagiert. Auch dafür kann eine

bestehende Datenbindung verwendet werden. Bei der Angabe der Datenbindung kann die Eigenschaft *Mode* eingestellt werden, die die Bindungsrichtung zwischen Steuerelement und Datenobjekt festlegt. Folgende Modi können eingestellt werden.

```
<TextBox Computername="{Binding Computername, Mode=OneTime}" />
<TextBox Computername="{Binding Computername, Mode=OneWay}" />
<TextBox Computername="{Binding Computername, Mode=TwoWay}" />
```

Durch das Setzen der Eigenschaft ist die Bindungsrichtung vorgegeben. Im Datenobjekt müssen die Änderungen aber noch veröffentlicht werden, damit die WPF die Daten an das Steuerelement weiterreichen kann. Zum Veröffentlichen einer Änderung im Datenobjekt muss die Schnittstelle *INotifyPropertyChanged* definiert werden. Nachfolgendes Beispiel zeigt anhand einer Klasse *Person* die Implementierung. Die Klasse enthält zwei Eigenschaften *Nachname* und *Vorname* deren Änderungen überwacht werden.

```
public class Person : INotifyPropertyChanged {

// Event aus INotifyPropertyChanged
public event PropertyChangedEventHandler PropertyChanged;

// Felder
private string _nachName;
private string _vorName;

//Getter und Setter
public string NachName {
  get { return this._nachName; }
  set {
    this._nachName = value;
    OnPropertyChanged( "NachName" );
  }
}

public string VorName {
  get { return this._vorName; }
  set {
    this._vorName = value;
    OnPropertyChanged( "VorName" );
  }
}
// On-Methode für den Aufruf des Ereignisses
protected virtual void OnPropertyChanged( string propertyName ) {

if ( PropertyChanged != null ) {
  PropertyChanged( this, new PropertyChangedEventArgs(propertyName) );
}
...
```

4.3.3.3 Listview an das Ergebnis einer SQL-Abfrage binden

In datenorientierten Anwendungen kommen die Daten die über die Steuerelemente angezeigt werden üblicherweise aus einer Datenbank. Mit den Techniken der Datenbindung und Datenkontext, die die WPF zur Verfügung stellt, ist dies relativ einfach realisierbar. Anhand eines Beispiels werden die Techniken erläutert. Ein Steuerelement der Klasse *Listview* soll die Daten folgender einfacher SQL-Abfrage anzeigen:

```
SELECT DISTINCT Computername, UserID, Scandatum  from vData;
```

Zuerst wird in XAML das entsprechende Listview-Element definiert. Das Listview-Element enthält 3 Spalten (*GridViewColumn*) die an die jeweiligen Spaltennamen der SQL-Abfrage gebunden wurden:

```
<ListView Name="listView2" ItemTemplate="{DynamicResource
SoftwareTemplate}" ItemsSource="{Binding Path=Table}"
 <GridView>
  <GridViewColumn
   Header="Auswahl Computer" DisplayMemberBinding="{Binding
                             Path=Computername}">
  </GridViewColumn>
  <GridViewColumn
   Header="zugehöriger Benutzer" DisplayMemberBinding="{Binding
                             Path=UserID}" >
  </GridViewColumn>
  <GridViewColumn
   Header="Scandatum" DisplayMemberBinding="{Binding Path=Scandatum}">
  </GridViewColumn>
 </GridView>
</ListView>
```

Damit im Listview jetzt die eigentlichen Daten angezeigt werden, muss dem Datenkontext noch die Datenquelle zugewiesen werden. Die Anbindung erfolgt über den C#-Code:

```
SqlConnection con = new SqlConnection("server=...");
con.Open();

DataSet ds = new DataSet();
string strAbfrage = "SELECT DISTINCT Computername, UserID, Scandatum
from vData ";
SqlDataAdapter ad = new SqlDataAdapter(strAbfrage, con);
ad.Fill(ds);
listView2.DataContext = ds.Tables[0].DefaultView;
```

Das Ergebnis sieht wie folgt aus:

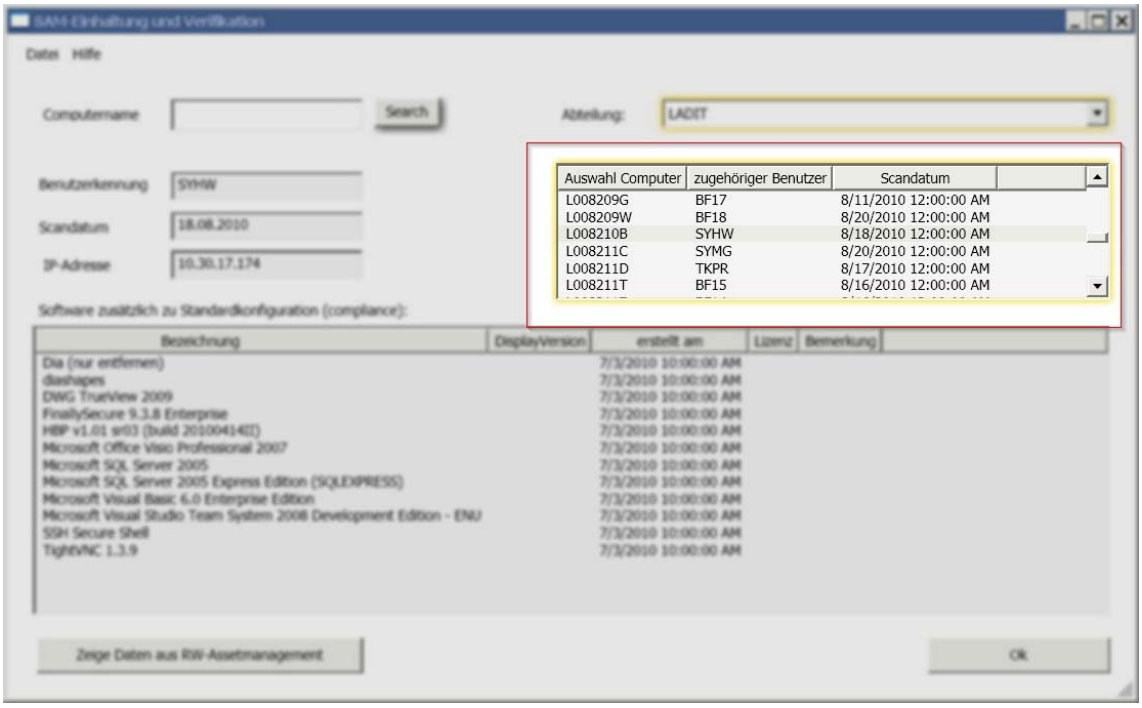


Abbildung 26: Listview an eine SQL-Abfrage gebunden

4.3.3.4 Fazit

Mit der eingeführten Techniken der Datenbindung in WPF hat Microsoft ein sehr weit reichendes Konzept geschaffen. Die Tatsache, dass Datenbindung jetzt auf viele Eigenschaften (Dependeny Property) angewendet werden kann zeigt die umfassenden Möglichkeiten die hier geschaffen wurden. In der Literatur (vgl. [Eller, 2008], [Wegener, 2009]) wird aber größtenteils auf die grafischen Fähigkeiten eingegangen. Dies könnte dazu führen das es von den Entwicklern von Informationssystemen nicht angenommen wird. Auch ist der Aufwand des Erlernens einer zusätzlichen Beschreibungssprache (XAML) nicht zu unterschätzen. Grundsätzlich kann festgehalten werden, dass WPF für datenorientierte Anwendungen geeignet ist. Die Entwicklungszeit gegenüber Windows-Forms-Anwendungen ist aber derzeit noch deutlich höher.

4.3.4 Darstellung der Informationsinhalte und Ihre Datenbankzugriffe

4.3.4.1 Zugriff auf die SAM-Datenbank - View vData

Um den Zugriff auf die Daten der SAM-Datenbank möglichst einfach zu halten wird eine View *vData* definiert. Über die View erhält man Zugriff auf alle Clients, die zugeordnete Software und wer zuletzt auf den Client eingestiegen ist. Abbildung 27 zeigt die Definition der View.

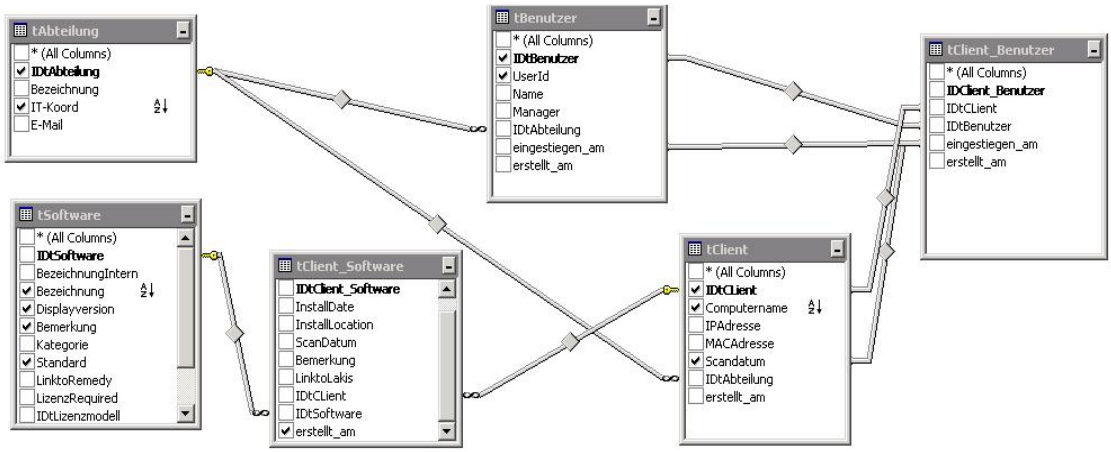


Abbildung 27: Definition der View vData

Über diese View kann mit einer SQL-Abfrage die installierte Software eines Clients abgefragt werden:

Select * from vData where Computername = ‚L008210B’ and standard = 0

4.3.4.2 Zugriff auf RW-Assetmanagement

Der Zugriff auf das RW-Assetmanagement erfolgt ausschließlich über Views. Diese Views werden durch das Produkt bei der Erstellung der Bearbeitungsmasken erstellt und können zur Abfrage der Daten genützt werden. In der Anlage 7 ist die Definition der Views als SQL-Skript dargestellt.

- View - NOE_Asset

Über die View NOE_Asset werden alle Assets die im RW-Assetmanagement definiert sind zur Verfügung gestellt. Für den Abgleich mit der SAM-Datenbank kann die Spalte *Computername* herangezogen werden. Es daher möglich mit SQL-Abfragen zu ermitteln ob ein Client der über den Softwarescan gefunden wurde auch im RW-Assetmanagement inventarisiert ist.

- View – NOE_SWLizenzkauf

Über diese View kann auf alle definierten SW-Assets zugegriffen werden. Es sind derzeit die lizenzpflichtigen Softwareprodukte, die seitens der Abteilung LAD1-IT angeschafft wurden, erfasst. Um eine Beziehung zwischen der gescannten Software und der erfassten Software herstellen zu können muss in der Tabelle tSoftware in der Spalte *LinktoRemedy* der Produktcode der im RW-Assetmanagement vergeben wurde erfasst werden.

- View – NOE_SWAsset_HWAsset

In dieser View ist die Zuordnung zwischen SW-Asset und HW-Asset abgespeichert. Es kann damit festgestellt werden, ob einem HW-Asset ein SW-Asset zugeordnet ist.

4.3.5 Berechtigungskonzept

Das Berechtigungskonzept der Clientanwendung ist so aufgebaut, das das Rollenkonzept aus 3.4.1.1 weitgehend ohne administrativen Zusatzaufwand abgebildet werden kann. Die Berechtigungen können aus dem Active Directory (AD) der NÖL ausgelesen und mit den Daten die aus der Clientscananwendung geliefert werden zugeordnet werden. In Abbildung 28 ist ein Benutzer im Active Directory der NÖL dargestellt.

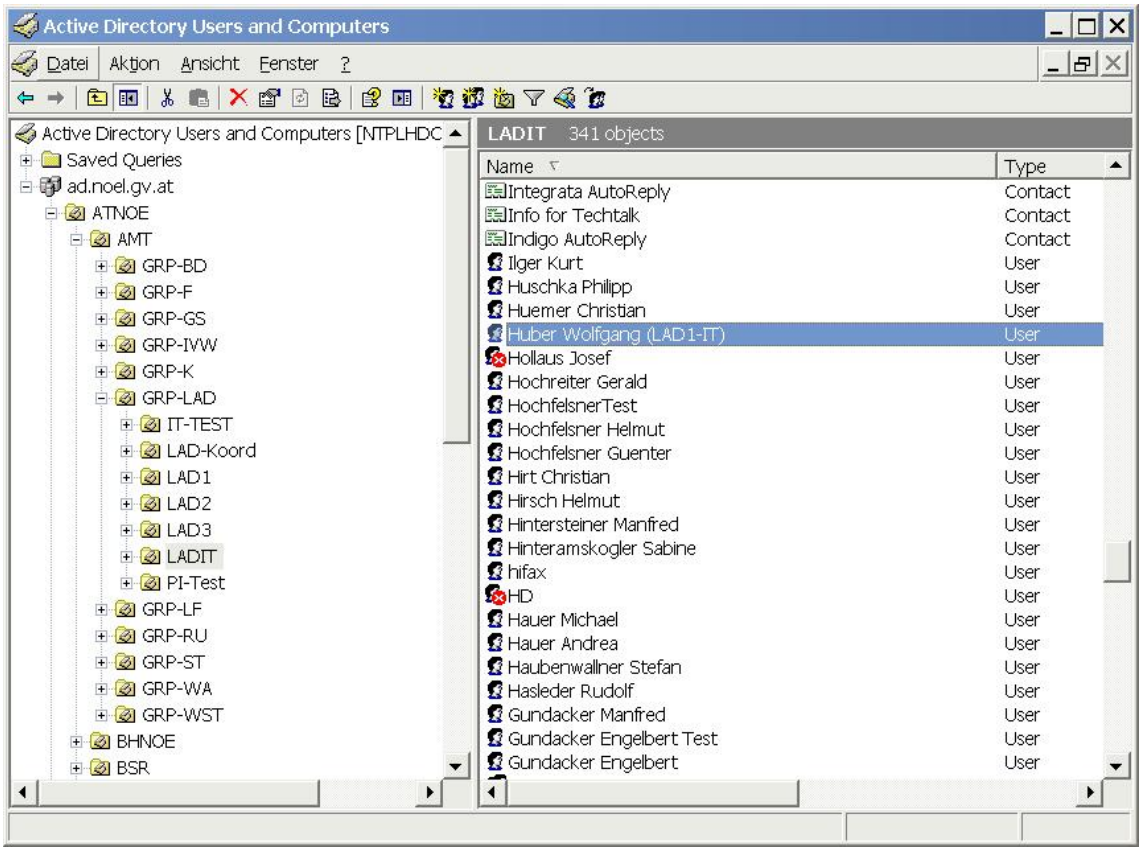


Abbildung 28: Benutzer im Active Directory der NÖL

In der Tabelle *tAbteilung* ist in der Spalte *IT-Koord* der Name einer AD-Gruppe eingetragen. Mitglieder dieser AD-Gruppe haben Zugriff auf die der Abteilung zugeordneten Clients. Die Tabelle *tClient* ist mit dem Fremdschlüssel *IDtAbteilung* mit der Tabelle *tAbteilung* verknüpft. Über diese Beziehung können alle zugeordneten Clients ermittelt

werden. Abbildung 29 zeigt die Ermittlung der Berechtigung. Es werden aus dem AD alle Mitgliedschaften ausgelesen die mit „-Koord“ enden die dem eingestiegenem Benutzer zugeordnet sind. Über diese Mitgliedschaften kann ermittelt werden, auf welche Abteilungen ein Benutzer Zugriff hat. Über die Abteilung können dann die anzuzeigenden Clients ausgelesen werden.

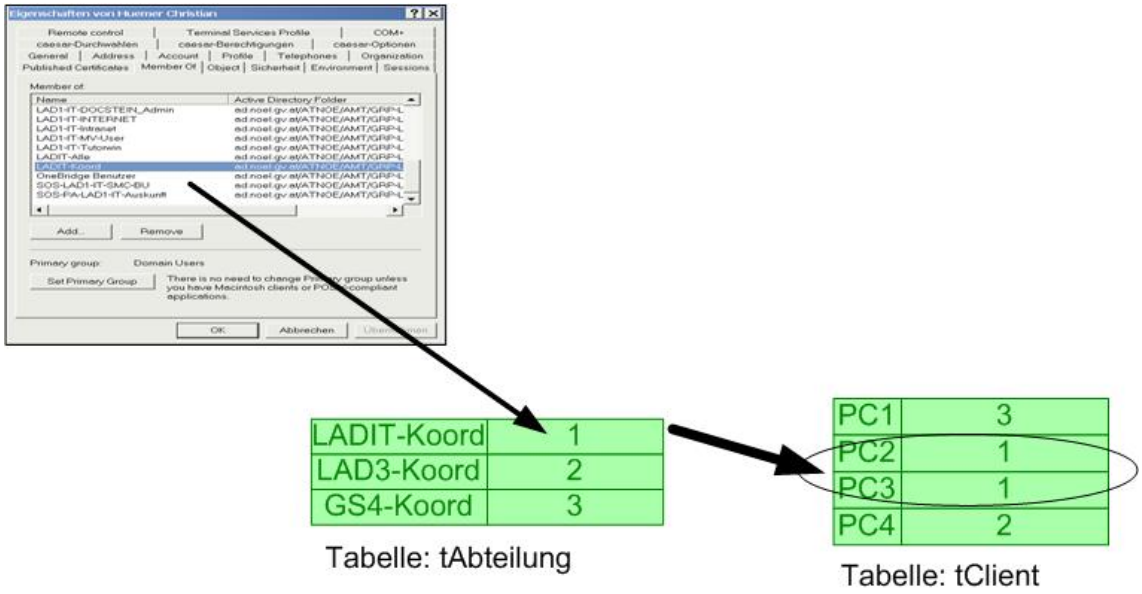


Abbildung 29: Berechtigungsprüfung

Mit dieser Berechtigungsprüfung können die Rolle IT-Koordinator und Administrator abgedeckt werden. Die Rolle Benutzer muss gesondert behandelt werden. Ist ein Benutzer kein Mitglied einer AD-Gruppe die mit „-Koord“ endet, so darf er nur die Daten des Clients sehen auf dem er gerade eingestiegen ist.

4.4 Systemkonzept Serveranwendung

4.4.1 Allgemeines

Die Aufgaben der Serveranwendung gliedern sich grundsätzlich in zwei Bereiche:

- Empfang und Abspeichern der XML-Dateien und
- Einpflege der XML-Dateien in die Datenbank

Um die Anwendung ausfallsicher auszulegen wird darauf geachtet, dass alle Anwendungsteile redundant ausgelegt werden. Abbildung 30 zeigt den redundanten Ansatz der Implementierung.

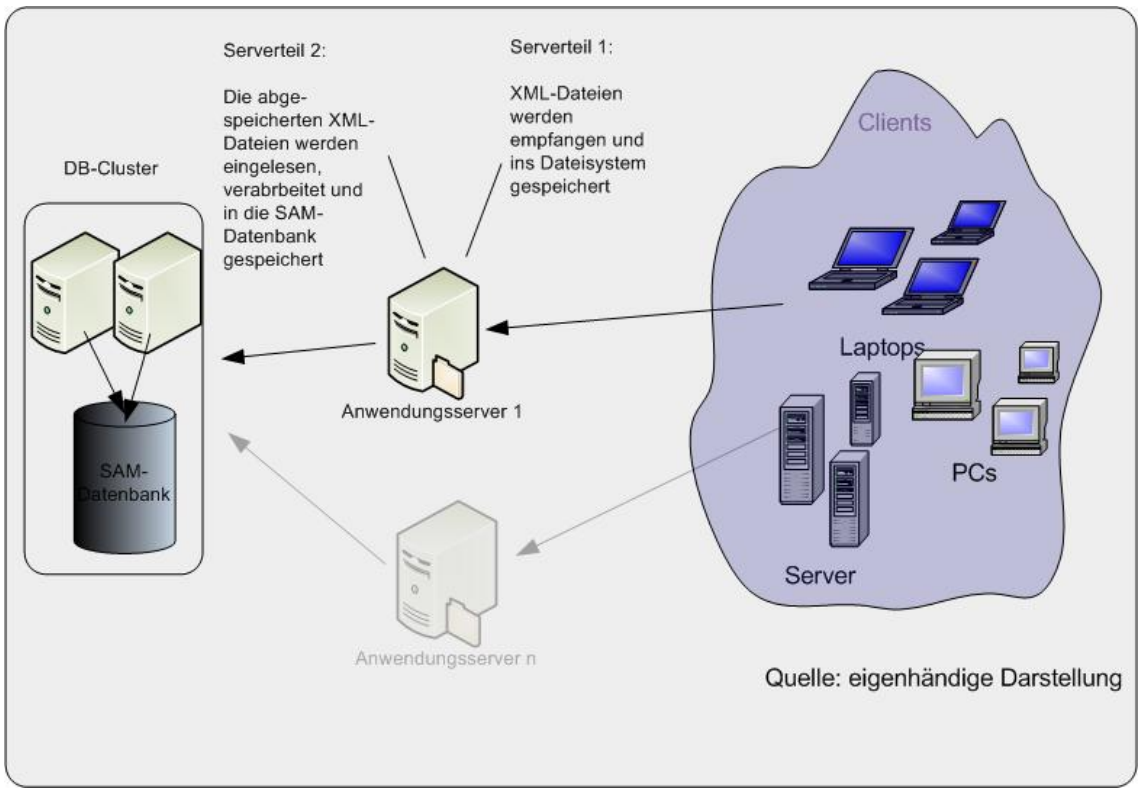


Abbildung 30: Architekturbild Serveranwendung

Der erste Teil der Serveranwendung ist für den Empfang der Daten der Clients zuständig. Die Clients schicken die Daten, wie in Kap. 4.2.3 beschrieben, mittels WinSock zum ersten Anwendungsserver zu dem die Socketkommunikation etabliert werden kann. Das Service schreibt die empfangenen Daten als Datei in ein Verzeichnis im lokalen Dateisystem. Dadurch, dass das Clientscanprogramm mehrere Anwendungsserver bzw. Anwendungsservices konfiguriert haben kann, kann damit auf einfache Art Ausfallsicherheit gewährleistet werden und eine Lastverteilung stattfinden. Der zweite Teil der Serveranwendung ist der datenbankorientierte Teil. Hier werden die abgespeicherten XML-Dateien gelesen und die Daten entsprechend in der Datenbank abgespeichert. Danach kann eine verarbeitete XML-Datei entweder gelöscht oder zur Nachverfolgung in ein Archivverzeichnis verschoben werden.

4.4.2 Anwendungsservice 1: Empfang und Abspeichern der XML-Dateien

Wie schon in Kap. 4.1.1 beschrieben handelt es sich um eine kompakte Anwendung die den empfangenen Datenstrom eines Clients auf das lokale Dateisystem ab-

speichert. Die Funktion ist, in vereinfachter Form, als Flussdiagramms (Abbildung 31) dargestellt.

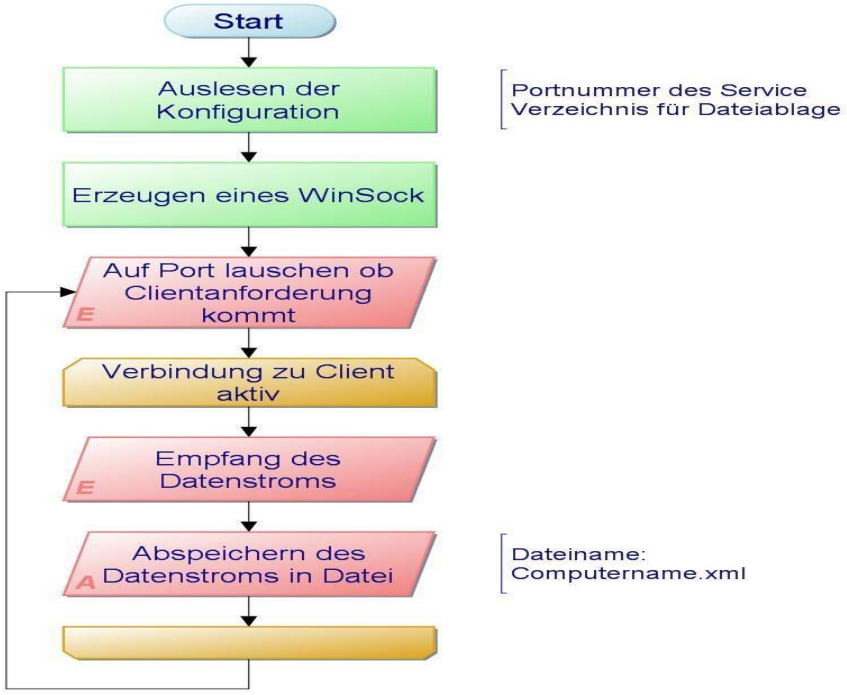


Abbildung 31: Flussdiagramm Anwendungsservice1

4.4.3 Anwendungsservice 2: Verarbeitung der XML-Dateien in die Datenbank

Nachdem die Daten der Clients lokal im Dateisystem abgelegt wurden werden in regelmäßigen Intervallen die Dateien in die Datenbank importiert. Der Aufruffrequenz soll über den eingebauten Dienst „Geplante Tasks“ in Windows 2008 Server gesteuert werden. Die Verarbeitungslogik ist so gestaltet worden, dass der Import der Daten im Stundenintervall gestartet werden kann. Der Datenbankzugriff erfolgt über ADO.Net dazu gibt Kapitel 4.4.3.1 einen kurzen Überblick über das Grundkonzept. Die grundlegende Ablauflogik der Verarbeitung ist in den Abbildungen 31 und 32 dargestellt.

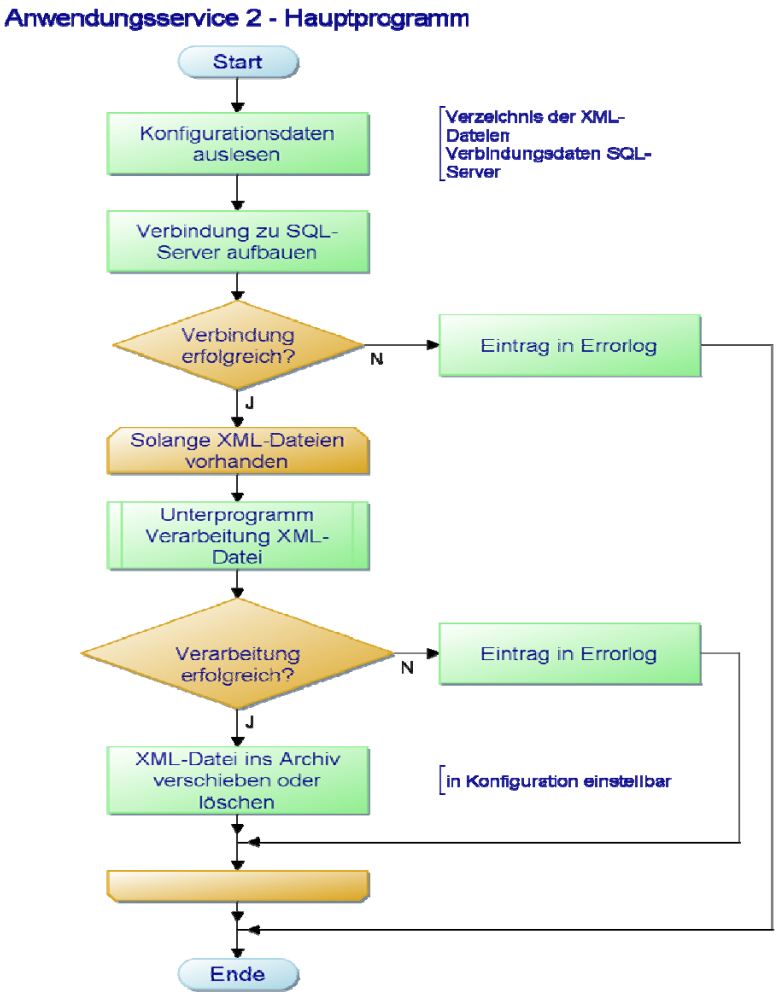


Abbildung 32: Flussdiagramm Anwendungsservice 2

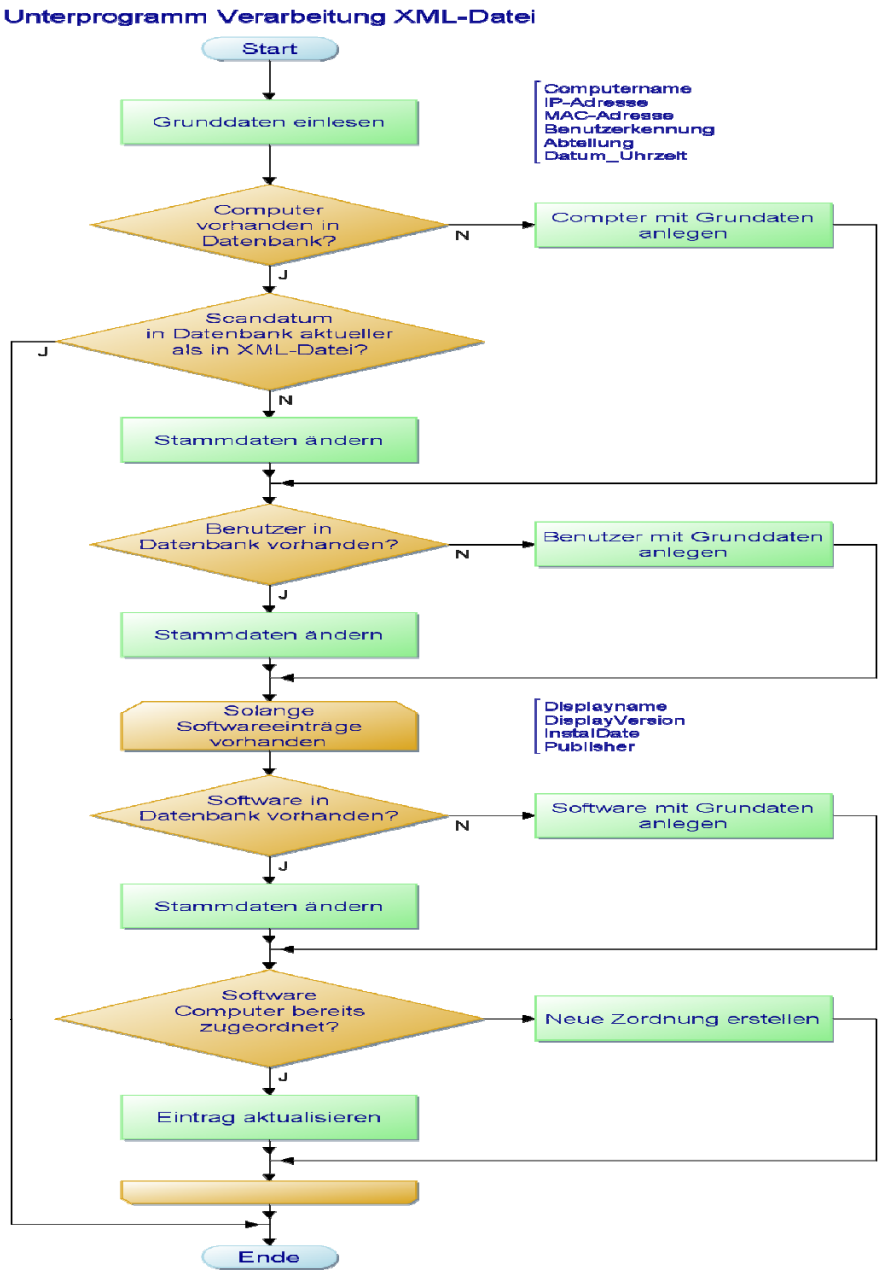


Abbildung 33: Flussdiagramm Anwendungsservice 2 – Datenbankaktualisierung

4.4.3.1 Zugriff auf die Datenbank über ADO.Net

Der Zugriff auf Datenbanken erfolgt über SQL-Kommandos. Wenn eine Anwendung eine Datenbank nutzen will, so muss sie diese Kommandos absetzen können. Im .NET Framework ist hierfür der Namespace System.Data vorhanden. Dieser beinhaltet die Klassen die benötigt werden, um auf Datenbanken komfortabel zugreifen zu können. Die Sammlung dieser Klassen wird als ADO.NET bezeichnet. Abbildung 34 zeigt die Komponenten von ADO.NET.

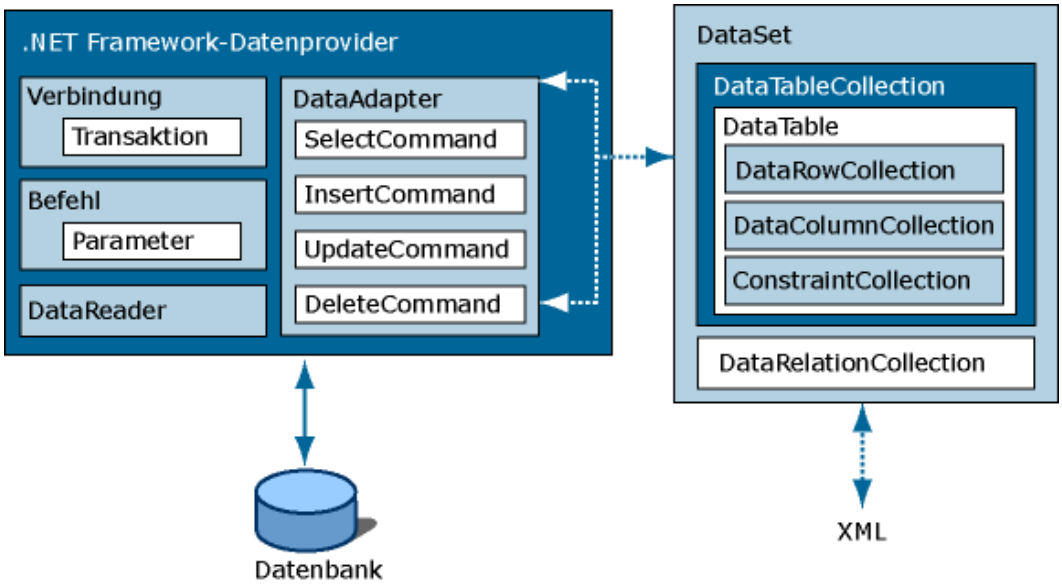


Abbildung 34: ADO.NET Architektur [16]

Folgend werden die wichtigsten Klassen erläutert die nötig sind, um die notwendigen Datenbankmanipulationen durchführen zu können. Der Zugriff auf eine Datenbank erfolgt immer nach dem gleichem Schema:

- Aufbau der Verbindung zur Datenbank,
- SQL-Kommandos durchführen,
- Ergebnisse behandeln (bei Select-Anweisungen),
- Verbindung zur Datenbank schließen.

Verbindungsaufbau

Für den Verbindungsaufbau ist die Klasse SqlConnection zuständig. Der Klasse wird eine sogenannte Verbindungszeichenfolge (engl. Connectionstring) übergeben. Der Connectionstring enthält alle Informationen die zum Aufbau der Verbindung notwendig sind. Die wichtigsten Parameter sind:

- Data Source
  Gibt die Datenquelle an, z.B.: den Servernamen oder „(local)“ für den lokalen Server
- Integrated Security
- User Id
  Benutzername mit der sich zur Datenbank verbunden wird
- Passwort
  Passwort das zu Benutzername vergeben ist

[16] Bildquelle: [ADO.NET, 2010]

- Initial Catalog
  Entspricht der Datenbank die angesprochen werden soll

4.5 Systemkonzept Datenbank

4.5.1 Allgemeines

Die Datenbank des SAM - Systems dient zur Aufnahme aller notwendigen Informationen die zur Durchführung des Prozesses notwendig sind. Wie in den systembezogenen Anforderungsmerkmalen definiert wird als Datenbankmanagementsystem Microsoft SQL Server 2008 eingesetzt. Dieser ist als Cluster ausgeführt (siehe Anlage 5). Im Folgenden wird speziell auf ausgewählte Funktionalitäten die benutzt werden eingegangen.

4.5.2 Eingesetzte Funktionalitäten

4.5.2.1 Authentifizerung

SQL Server 2008 bietet zwei Authentifizierungsmethoden an:

- Integrierte Windows Authentifizierung

Bei der integrierten Windows Authentifizierung wird dem SQL Server kein Benutzername/Kennwort übergeben, sondern er verlässt sich auf die Anmeldung die zuvor auf dem System des aufrufenden Clients in der Windows Domäne stattgefunden hat. Der Vorteil liegt darin, dass Benutzer nicht am SQL-Server direkt angelegt werden müssen. Für die Benutzer oder Gruppen in der Windows-Domäne gelten unternehmensweite Sicherheitsrichtlinien (Kennwortlänge, komplexe Kennwörter, …)

- SQL Server Authentfizierung

Bei der SQL Server Authentifizierung werden eine Benutzerkennung und ein Kennwort übermittelt. Der SQL Server überprüft die Konteninformation gegen die eigene Benutzerdatenbank. Der Vorteil liegt darin, dass keine Windows-Domäne notwendig. Als Nachteil ist zu sehen, dass die Benutzer am SQL Server angelegt werden müssen und keine unternehmensweite Sicherheitsrichtlinie gilt

Für den Zugriff auf die SAM-Datenbank kommt die SQL Server Authentifizierung zum Einsatz. Dies hat den Vorteil, dass die Anwendungsprogramme am Server ohne Anmeldung eines Benutzers gestartet werden kann.

4.5.2.2 Change Data Capture

Wie in Kap. 3 definiert ist für gewisse Datenfelder zu protokollieren wenn sich der Inhalt ändert. Mit Change Data Capture (CDC) ist in SQL Server 2008 diese Funktionalität neu hinzugekommen. Eine Besonderheit ist, dass die Lösung auf dem Transaktionsprotokoll des SQL Servers basiert und nicht auf Basis von Triggern. Vorteile liegen hier in der verbesserten Performance. Weiters kann ein „schlecht" programmierter Trigger die eigentliche Datenbankaktion dadurch nicht mehr beeinflussen. Abbildung 35 zeigt die Architektur der neuen Funktion. Diese Funktion wird beim Datenbankdesign für die Tabellen (tAbteilung, tClient, tBenutzer) genutzt. Im Kapitel 5.1 wird auf die Aktivierung eingegangen.

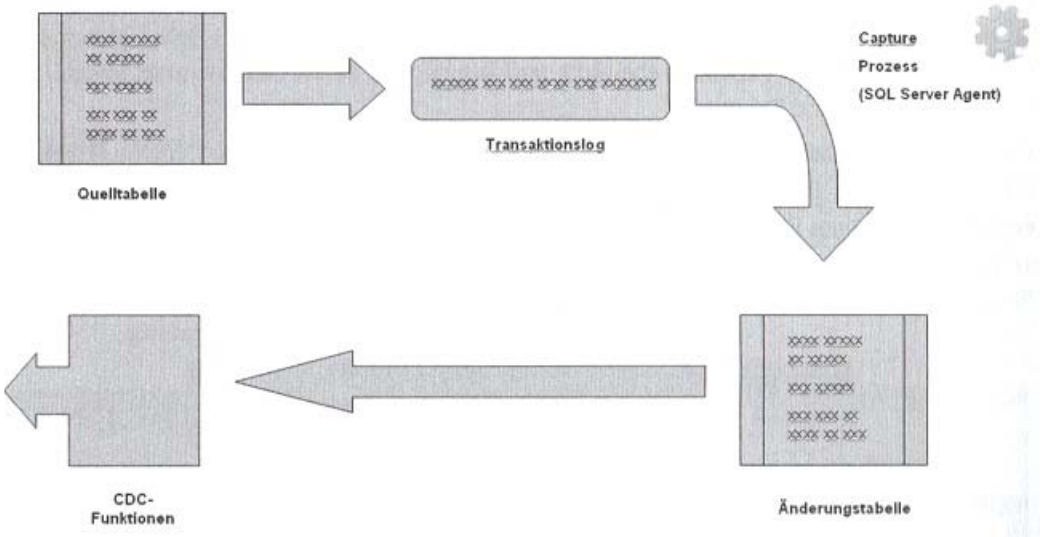


Abbildung 35: Architektur Change Data Capture(CDC)[17]

[17] Bildquelle: [Kansy, 2008] S. 62

4.5.3 Datenbankdesign

4.5.3.1 Allgemeines

Das Datenmodell beschreibt die Informationsgruppen und ihre Beziehungen zueinander. Für den Entwurf des Datenbankmodells wird das Entity-Relationship-Modell (ER-Modell) verwendet. Nach ([WIKI01, 2010], ist das ER-Modell wie folgt definiert:

> „Das **Entity-Relationship-Modell**, kurz ER-Modell oder **ERM**, deutsch **Gegenstands-Beziehungs-Modell**, dient dazu, im Rahmen der semantischen Datenmodellierung einen Ausschnitt der realen Welt zu beschreiben. Das ER-Modell besteht aus einer Grafik (siehe Abbildung 36) und einer Beschreibung der darin verwendeten Elemente, wobei Dateninhalte (d.h. die Bedeutung bzw. Semantik der Daten) und Datenstrukturen dargestellt werden."

1976 wurde von Peter Chen das ER-Modell erstmals veröffentlicht. Das ER-Modell ist auch bei der NÖL der Standard für die Datenmodellierung. Bei der Darstellung des ER-Modells wird die Chen-Notation (vgl. [WIKI01, 2010]) verwendet. Abbildung 36 zeigt ein Beispiel für ein ER-Modell in Chen-Notation.

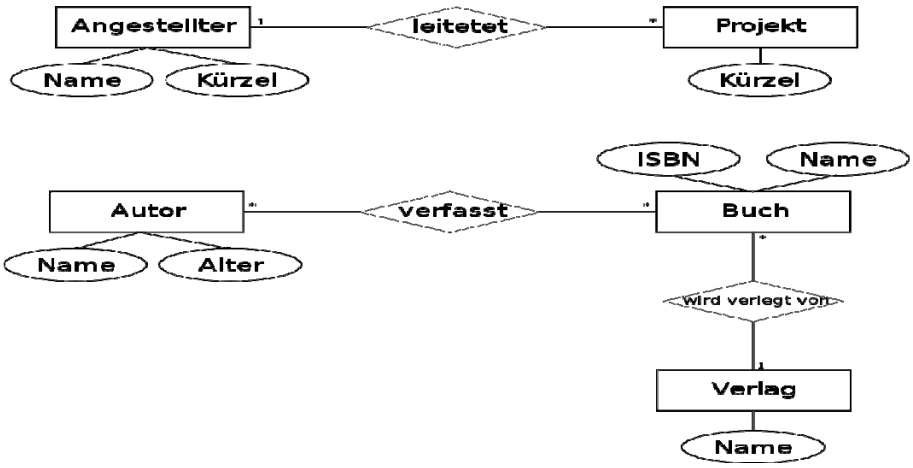


Abbildung 36: Beispiel für ER-Modell in Chen-Notation[18]

Die Rechtecke stellen die Entitätsklassen, die Rauten die Beziehungsmengen zwischen den Entitätsklassen und die Ovalen die Attribute der Entitätsklassen dar. Es existieren aber auch weitere Notationen wie die Martin-Notation, Bachman-Notation oder UML (vgl. [Wiki02, 2010]).

[18] Bildquelle: [WIKI01, 2010]

Die Darstellung der Kardinalitäten ist in [WIKI02, 2010] wie folgt definiert:

> „1:1 (lies [0 oder 1] zu [1 oder 0])
> Jede Entität aus der ersten Entitätsmenge kann mit höchstens einer Entität aus der zweiten Entitätsmenge in Beziehung stehen, und umgekehrt.
>
> 1:n (lies [0 oder 1] zu beliebig vielen)
> Jede Entität aus der ersten Entitätsmenge kann mit beliebig vielen Entitäten aus der zweiten Entitätsmenge in Beziehung stehen. Jede Entität aus der zweiten Entitätsmenge kann mit höchstens einer Entität aus der ersten Entitätsmenge in Beziehung stehen.
>
> m:n (lies beliebig viele zu beliebig vielen)
> Jede Entität aus der ersten Entitätsmenge kann mit beliebig vielen Entitäten aus der zweiten Entitätsmenge in Beziehung stehen, und umgekehrt. „

4.5.3.2 Entity-Relation Modell

Das ER-Diagramm in Abbildung 37 soll veranschaulichen, wie das Datenbankmodell aufgebaut werden soll. Es werden in dem Modell die wichtigsten Relationen dargestellt Die komplette Datenbank ist im Kap. 5.1 und in Anlage 6 dargestellt. Hier kann anhand der SQL-Skripte die komplette Datenbank nachvollzogen werden. Für die Darstellung des ER-Modells wurde auf die freie Software Dia[19] zurückgegriffen. Die Software erlaubt es ähnlich wie bei kommerziellen Programmen, wie Microsoft Visio anhand von Vorlagen Diagramme zu zeichnen.

[19] Anwendung zum Zeichnen strukturierter Daten, http://live.gnome.org/Dia

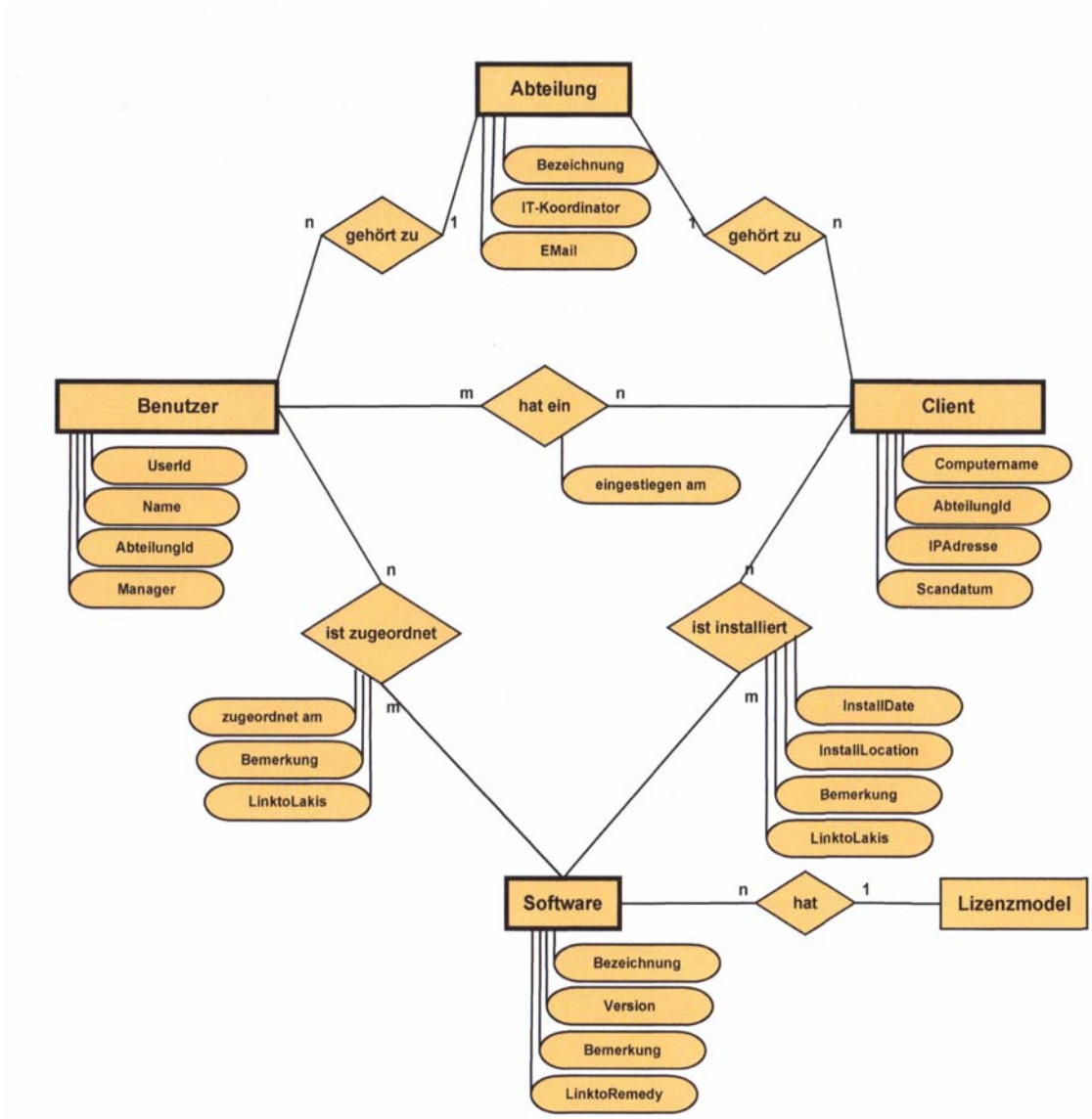


Abbildung 37: Entity-Relationship Diagramm

Basis des Datenmodells bildet das Gerät das gescannt wird. Das Gerät wird mit seinen Grunddaten Computername, IP-Adresse, MAC-Adresse und Abteilung welche lokal ausgelesen werden aufgenommen. Über den Computernamen kann der Abgleich mit den Geräten die in der Anwendung „RW-Assetmanagement“ erfasst wurden vorgenommen werden. Bei jedem Scan wird die installierte Software mit dem Gerät verknüpft. Neu hinzukommende Software wird in der Entität Software eingetragen. Jede Software wird über Ihre Bezeichnung in die Tabelle Software aufgenommen. Als Zusatzinformation kann ein Verweis zum „RW-Assetmanagement“ eingetragen werden.

4.5.3.3 Datenmengen

Daten:	Clients
Häufigkeit:	~ 10.000
Beschreibung:	Zu den Clients gehören z.B. PC's, Laptops, Server.

Daten:	Benutzer
Häufigkeit:	~ 6.500
Beschreibung:	Jeder Mitarbeiter der von der NÖL eine Benutzerkennung zugewiesen hat.

Daten:	Software
Häufigkeit:	unbegrenzt
Beschreibung:	Jede eindeutig identifizierte Software

Daten:	Abteilung
Häufigkeit:	~ 200
Beschreibung:	Jeder Benutzer und jedes Gerät ist genau einer Abteilung zugeordnet.

Daten:	Lizenzmodell
Häufigkeit:	wenige
Beschreibung:	Beschreibung eines Lizenzmodells, dass einer Software zugeordnet ist.

Daten:	Zuordnung Benutzer - Client
Häufigkeit:	unbegrenzt
Beschreibung:	Hier wird jeder Einstieg eines Benutzers auf einem Client registriert

Daten:	Zuordnung Client - Software
Häufigkeit:	unbegrenzt
Beschreibung:	Beinhaltet die dem jeweiligen Gerät zugeordnete Software. Erwartet werden circa 2 Millionen Datensätze.

Daten:	Zuordnung Benutzer - Software
Häufigkeit:	unbegrenzt
Beschreibung:	Beinhaltet die dem jeweiligen Benutzer zugeordnete Software.

5 Realisierung des Prototypen

Dieses Kapitel behandelt die eigentliche (technische) Umsetzung der Arbeit. Dazu wird zunächst die realisierte Funktionalität gezeigt. Anschließend werden die wichtigsten Details der Entwicklung anhand von Quellcodebeispielen vorgestellt.

5.1 Die Datenbank

Auf Basis des ER-Diagramms wurde die Datenbank entworfen. Die Nutzung der entworfenen Tabellen wird erläutert. Spalten mit spezieller Bedeutung werden dabei näher erklärt. Die Beschreibung aller Tabellen, Referenzen und Indizes nebst dazugehörigen Skripten ist im Anhang 6 enthalten. Abbildung 37 gibt grafisch einen Überblick über die entworfene Datenbank.

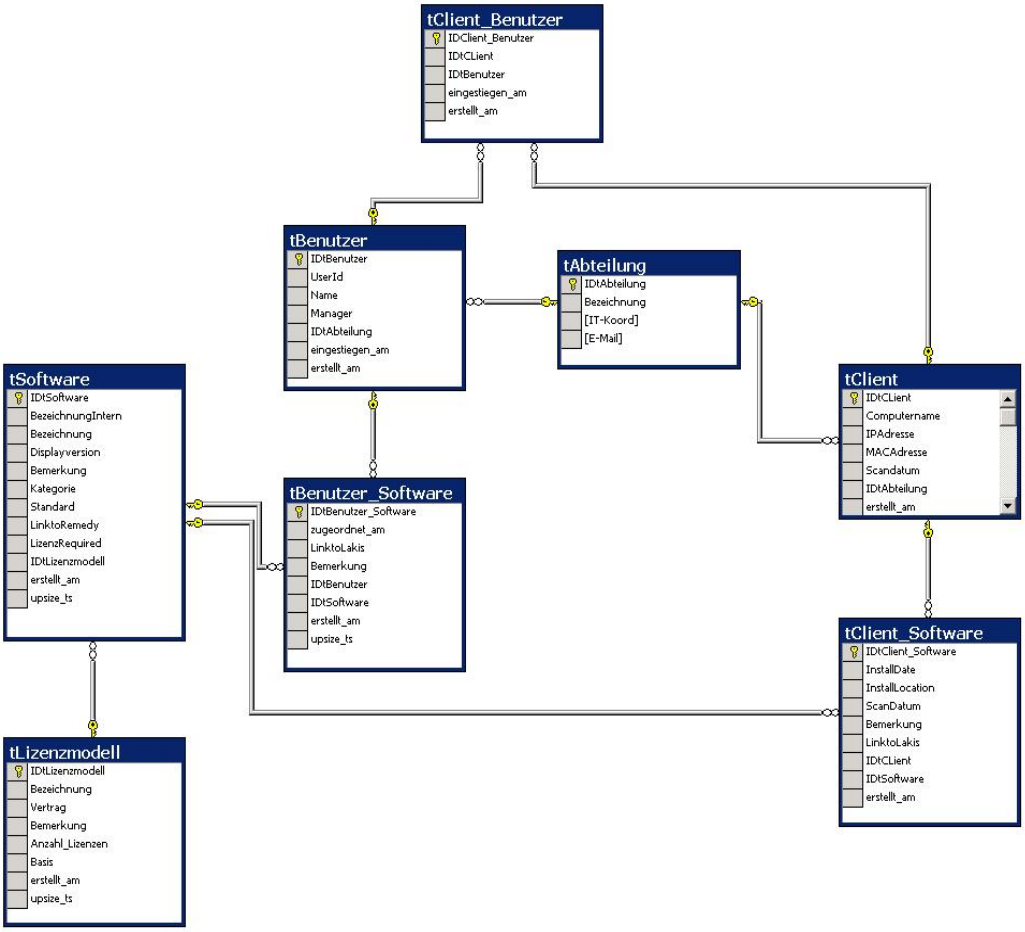


Abbildung 38: SAM-Datenbank

5.1.1 Die Stammdaten

5.1.1.1 Tabelle tSoftware

In der Tabelle *tSoftware* werden alle, durch das Clientscanprogramm, erfassten Softwareinstallationen (Software) gespeichert. Zwei Spalten besitzen eine besondere Bedeutung. Mit der Spalte *Standard* kann gesteuert werden ob eine Software zum Standardclient der NÖL gehört. Ist die Spalte *Standard* gesetzt (wahr) wird die Software in der Clientanwendung nicht angezeigt. Über die Spalte *LinktoRemedy* wird der Verweis zum RW-Assetmanagement hergestellt. Es wird der Schlüssel aus dem RW-Assetmanagement eingetragen, damit ein direkter Zugriff möglich ist.

5.1.1.2 Tabelle tBenutzer

In der Tabelle *tBenutzer* werden die Daten des angemeldeten Benutzers abgespeichert. Die Daten können weiters zum Abgleich mit dem Active Directory herangezogen werden um zum Beispiel Benutzer die schon länger nicht eingestiegen sind zu bereinigen.

5.1.1.3 Tabelle tClient

In der Tabelle *tClient* werden die Daten des gescannten Client eingetragen. Auch diese Daten können wieder zum Abgleich mit dem RW-Assetmanagement herangezogen werden.

5.1.1.4 Tabelle tAbteilung

Die Tabelle *tAbteilung* beinhalten alle Abteilungen mit Ihrer Bezeichnung. Die Spalte *IT-Koord* ist von besonderer Bedeutung, da über diesen Eintrag der gesamte Berechtigungsvorgang gesteuert wird. Zusätzlich ist die E-Mail Adresse für den automatischen Berichtsversand abgespeichert.*Tabelle tLizenzen*

In der Tabelle *tLizenzen* sind die verschiedenen Lizenzarten beziehungsweise Lizenzierungsvarianten abgespeichert. Diese Daten müssen händisch erfasst werden. Abbildung 39 zeigt einen Ausschnitt der bei der NÖL erfassten Daten.

dbo_tLizenzmodell : Tabelle

IDtLizenzmodell	Bezeichnung	Vertrag	Bemerkung	Anzahl	Basis
1	.		keine Lizenzzuordnung für diese Software	0	
2	Micorsoft E/A Lizenz	ON-LAD1-IT-99/1244	Microsoft Office XP an NÖL-Standard Client	7000	Client
3	Finally Secure Enterprise Edition Lizenz	ON-LAD1-IT-07/0244	Festplattenverschlüsselung	500	Client
5	Greenshot		Bildschirmkopien anfertigen	0	
6	OpenSource		freie Software unter OpenSource Lizenz	0	
7	UltraEdit Lizenz			25	Client
8	AutoCad Lizenz für Einzelplatz	ON-LAD1-IT-07/1578	Lizenz für AutoCad Einzeplatzinstallation	15	Client

Abbildung 39: Lizenzierungsarten

5.1.2 Die Bewegungsdaten

5.1.2.1 tClient_Benutzer

Die Tabelle *tClient_Benutzer* stellt die Beziehung zwischen Client und eingestiegenem Benutzer her. In der Spalte *eingestiegen_am* wird der Zeitpunkt des Einstiegs (entspricht dem *Scandatum*) festgehalten.

5.1.2.2 tBenutzer_Software

In die Tabelle *tBenutzer_Software* werden die dem Benutzer direkt zugeordnete Software eingetragen. Zusätzlich kann in der Spalte *LinktoLakis* ein Verweis auf die Anforderung eingetragen werden.

5.1.2.3 tClient_Software

Der größte Anteil der Softwareprodukte ist einem Client direkt zugeordnet. Durch die Daten des Clientscanprogrammes wird diese Tabelle automatisch gefüllt. Es wird automatisch der Zeitpunkt des Scans eingetragen. Zusätzlich kann wieder in der Spalte *LinktoLakis* der Verweis auf die Anforderung und in der Spalte *Bemerkung* eine Zusatzinformation eingetragen werden.

5.1.3 Protokollieren von Änderungen

Wie bereits im Systemkonzept beschrieben werden für die Tabellen *tAbteilung, TBenutzer* und *tClient* auf gewisse Spalten die Überwachung von Änderungen aktiviert. Die Überwachung soll auf jenen Spalten aktiviert werden, auf dem das Berechtigungskonzept basiert. Um die Überwachung durchführen zu können muss diese generell für die Datenbank aktiviert werden. Dies geschieht mit folgender Anweisung:

```
EXEC sys.sp_cdc_enable_db_change_data_capture
```

Danach kann entweder für eine ganze Tabelle oder für Spalten einer Tabelle die Ablaufverfolgung aktiviert werden. Dafür steht die gespeicherte Prozedur *sys.sp_cdc_enable_table_change_data_capture* zur Verfügung. Nachfolgendes Beispiel zeigt die Aktivierung für die Spalte *IT-Koord* der Tabelle *tAbteilung* :

```
EXEC sys.sp_cdc_enable_table_change_data_capture
   @sourcename_schema = ´dbo´,
   @source_name = ´tAbteilung´,
   @role_name = ´db_admin´,
   @captured_column_list = [IT-Koord]
```

Die weiteren Aufrufe sind in Anlage 6 (Datenbankdefinition in DDL) angeführt.

5.2 Clientanwendung

5.2.1 XAML-Definition des Hauptfensters

Das Hauptfenster der Clientanwendung wurde in XAML definiert. Es zeigt sich, dass mit wenig Aufwand eine funktionelle Anwendung zum Darstellen des Ist-Zustandes der Clients entwickelt werden kann. Ohne zusätzlichen Programmcode reagiert die Anwendung sauber auf Größenänderungen. Abbildung 40 zeigt den Prototypen, wie die ermittelten Daten dem Benutzer präsentiert werden. In der Abbildung 40 ist ein Benutzer mit der Berechtigung auf die Abteilung „LADIT“ eingestiegen. Es werden automatisch die zugeordenten Clients in einer Listenansicht dargestellt. Mit Klick auf einen Eintrag kann die zusätzlich installierte Software angezeigt werden.

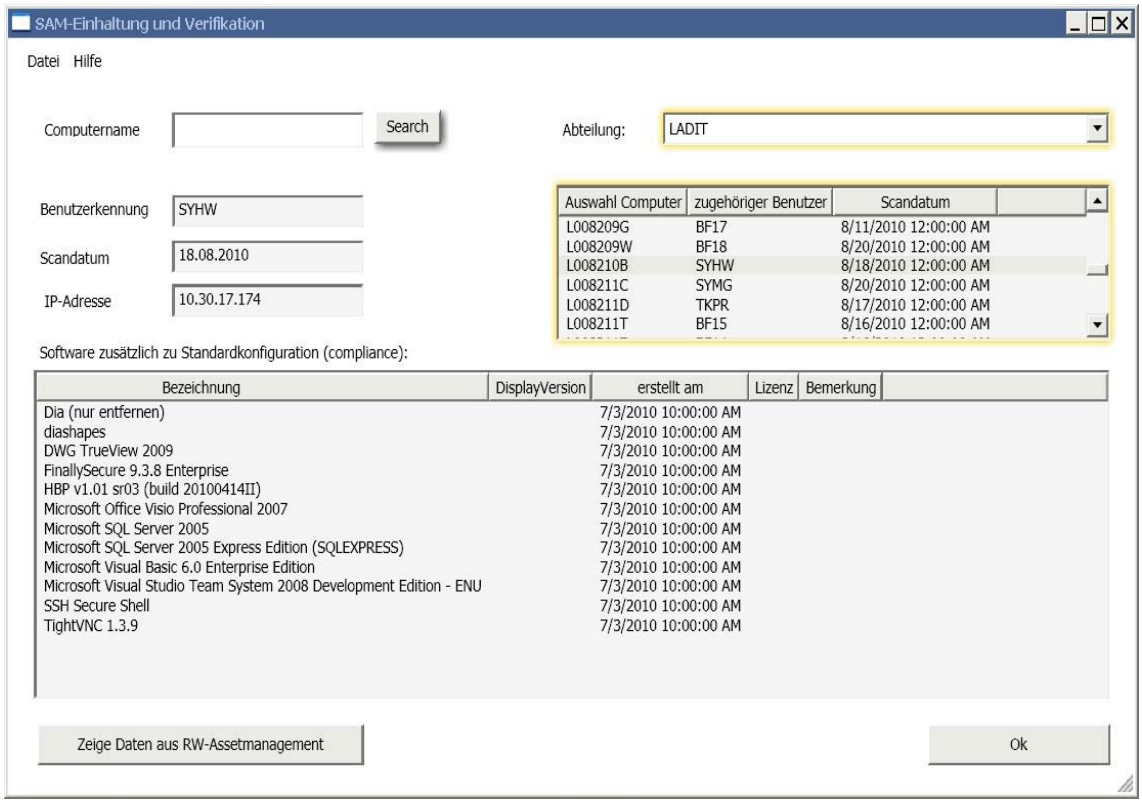


Abbildung 40: Hauptfenster der Clientanwendung

Folgender Programmausschnitt zeigt die XAML Definition des Hauptfensters. Man sieht hier sehr schön die Trennung der Darstellung von der Programmlogik. Bei den Steuerelementen ist das „Binding“ der Daten hinterlegt. Der eigentliche Datenkontext (die realen Daten) werden in der Programmlogik zugeordnet. Weiters sieht man die Zuordnung des jeweiligen Eventhandler wenn ein Ereignis über die Benutzeroberfläche ausgelöst wird. Anlage 8 zeigt die XAML-Definition des Hauptfensters.

```
<Window x:Class="SAMClientAnwendung.Window1"
  xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
  xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
  Title="SAM-Einhaltung und Verifikation" Height="681.803"
  Width="886.844" ResizeMode="CanResizeWithGrip" MinWidth="700"
...
   <ListView Margin="428.419,100,19,0" Name="listView2"
    MouseLeftButtonUp="listView2_MouseLeftButtonUp"
    ItemTemplate="{DynamicResource SoftwareTemplate}"
    ItemsSource="{Binding Path=Table}" Height="107.175"
    VerticalAlignment="Top" SelectedValuePath="Computername">
    <GridViewColumn
     Header="Auswahl Computer" DisplayMemberBinding="{Binding
     Path=Computername}">
    </GridViewColumn>
</Window>
```

5.2.2 Berücksichtigung der Rollen

Um auf Informationen im Active Directory zugreifen zu können steht im .NET Framework der Namespace System.DirectoryServices zur Verfügung. Damit hat man einen einfachen Zugriff auf die Objekte im Active Directory. Mit nachfolgendem Beispiel soll gezeigt werden, wie der Zugriff auf ein Benutzerobjekt erfolgt.

```
...
string Property = "cn";
DirectorySearcher findUsers = new DirectorySearcher();
findUsers.SearchRoot = domainRoot;
findUsers.Filter =
  "(&(objectClass=User)(objectCategory=person)(samAccountName=" +
  UserName +"))";
findUsers.PropertiesToLoad.Add(Property);
findUsers.Sort.PropertyName = Property;
findUsers.Sort.Direction = SortDirection.Ascending;
SearchResultCollection allEntries = findUsers.FindAll();
if (allEntries.Count == 1)
...
```

Über Zugriff auf die Tabelle *tAbteilung* können alle Abteilungen ausgelesen werden für die der eingestiegene Benutzer berechtigt ist. Anschließend wird das Ergebnis der Abfrage an den Datenkontext des Kombinationsfeldes *cboManGruppe* gebunden.

```
DataSet ds = new DataSet();
strAbfrage = "SELECT Bezeichnung, IDTAbteilung from tAbteilung ";
strWhere = " where [IT-Koord]IN (' + UserRights.ToString() +')";
string strOrderBy = " Bezeichnung ";
SqlDataAdapter ad = new SqlDataAdapter(strAbfrage + strWhere +
strOrderBy, con);
ad.Fill(ds);
cboManGruppe.DataContext = ds.Tables[0].DefaultView;
```

5.2.3 Daten in der Ergebnisliste

Bei Auswahl eines Clients über das Listenfeld oder über die Suche müssen die gescannten Softwareeinträge angezeigt werden. Durch die Definition des Standardclient der NÖL (*Standard = 0* in der SQL-Klausel) werden nur die zusätzlich installierten Softwarepackete angezeigt (siehe Abbildung 38). Für den Zugriff wird die vData verwendet. Die Ergebnisliste wird wieder dem Datenkontext des Listenfeldes für die Anzeige zugewiesen.

```
...
DataSet ds = new DataSet();
string strAbfrage;
strAbfrage = "SELECT Bezeichnung, Displayversion,  Bemerkung,
   Computername, erstellt_am, scandatum, Lizenz FROM vData ";
string strWhere;
string strComputer;
if (listView2.SelectedItem != null)
{
   ListView lv = sender as ListView;
   strComputer = lv.SelectedValue.ToString();
   strWhere = "WHERE Standard = 0 and Computername = '" + strComputer
      + "'";
   SqlDataAdapter ad = new SqlDataAdapter(strAbfrage + strWhere, con);
   ad.Fill(ds);
   listView1.DataContext = ds.Tables[0].DefaultView;
}
else
{
   listView1.DataContext = null;
}
...
```

5.3 Clientscanprogramm

Die Implementierung des Clientscanprogrammes als Systemprogramm muss nach den Vorgaben der NÖL in Microsoft Visual Basic 6.0 erfolgen. Durch die verschiedenen Entwicklungsumgebungen von Clientscanprogramm und Serveranwendung können sehr gut die Interoperabilität und die Möglichkeit der Abstimmung des Scanclients auf das jeweilige Betriebssystem gezeigt werden.

5.3.1 Grunddaten ermitteln

Wie im Systemkonzept definiert müssen als erstes die Grunddaten des zu scannenden Clients ermittelt werden. Die Ermittlung der Daten erfolgt einerseits über Umgebungsvariablen die bei der NÖL standardmäßig gesetzt sind, andererseits werden über API-Aufrufe Daten aus dem Betriebssystem ausgelesen. Im folgendem ist ein Ausschnitt dargestellt der die Daten ausliest.

```
strComputername = Environ$("COMPUTERNAME")
strAbteilung = Environ$("ABTEILUNG")
strFullname = Environ$("FULLNAME")
strBenutzer = fnGetHostUsId()
```

Der Computername, Abteilung und der Name des eingestiegenen Benutzers können über Umgebungsvariablen ausgelesen werden. Der Benutzername (UserID) des ein-

gestiegenen Benutzers wird über ein Windows-API ausgelesen. Nachfolgender Programmausschnitt zeigt die Deklaration der API-Funktion und die Nutzung in Visual-Basic.

```
Private Declare Function api_WNetGetUser Lib "mpr.dll" _
Alias "WNetGetUserA" (ByVal lpszLocalName As String, _
ByVal lpszUserName As String, lpcchBuffer As Long) As Long
Function fnGetHostUsId() As String
…
  strUserName = String(256, Chr$(0))
  lngResult = api_WNetGetUser(vbNullString, strUserName, 256)
  If lngResult = 0 Then
     strUserName = Left$(strUserName, InStr(1, strUserName, _
        Chr$(0)) - 1)
     fnGetHostUsId = Left$(UCase(strUserName), 25)
…
End Function
```

5.3.2 Installierte Software auslesen

Für das Auslesen der installierten Software muss der Schlüssel HKLM\ SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall ausgelesen werden. Zum Auslesen wurde auf die Implementierung der Windows Management Instrumentation (WMI) zugegriffen. Hier können mit einer Anweisung (EnumKey) alle Unterschlüssel in ein Array eingelesen und über eine Schleife abgearbeitet werden. Innerhalb der Schleife werden die einzelnen Werte ausgelesen. Falls ein Wert nicht vorhanden ist wird ein leerer Wert zurückgeliefert. Der folgende Programmausschnitt zeigt den Zugriff über WMI und die Schleife über die eingelesenen Unterschlüssel.

```
Const HKLM = &H80000002
Const KeyPath = "SOFTWARE\Microsoft\Windows\CurrentVersion\Uninstall"

Set objreg = GetObject(":\\" & strPC & "\root\default:StdRegProv")
objreg.EnumKey HKLM, KeyPath, valuearray  'Auslesen der Unterschlüssel

For Each value In valuearray
  On Error Resume Next
  objreg.GetStringValue HKLM, KeyPath & "\" & value, ValueName,_
      strDisplayname
  If Not IsNull(strDisplayname) Then
     If Not (InStr(1, strDisplayname, "KB") > 0) Then   'kein Hotfix
```

```
        objreg.GetStringValue HKLM, KeyPath & "\" & value, _
           "DisplayVersion", strDisplayVersion
        objreg.GetStringValue HKLM, KeyPath & "\" & value, _
           "InstallDate", strInstalldate
        objreg.GetStringValue HKLM, KeyPath & "\" & value, _
           "Publisher", strpublisher
...
 Next
```

5.3.3 Erstellen der XML-Datei

Für das Erstellen der XML-Datei wurde in das Visual Basic Project die Referenz „Microsoft XML, v6.0) eingebunden. Abbildung 41 zeigt die eingebundenen Referenzen des Visual Basic Projects.

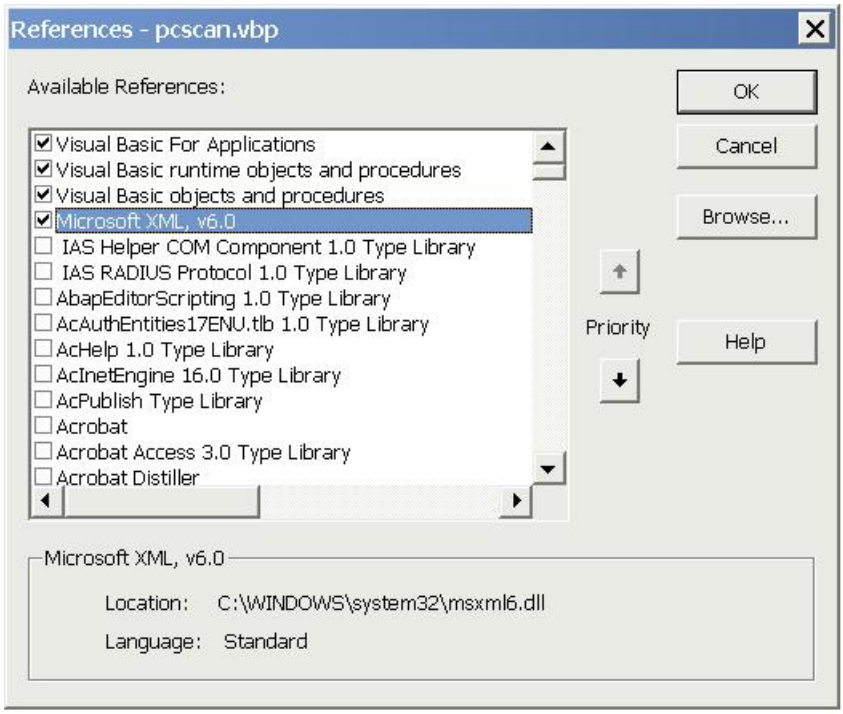


Abbildung 41: Eingebundene Referenzen des Clientscanprogrammes

Durch die Einbindung stehen Funktionen zum Erstellen von XML-Dateien zur Verfügung. Als erstes wird ein neues Dokument erzeugt und die XML-Deklaration eingefügt.

```
Set oDOM = New DOMDocument
Set oInstruct = oDOM.createProcessingInstruction("xml", _
    "version=""1.0""")
Call oDOM.insertBefore(oInstruct, oDOM.childNodes.Item(0))
```

Danach müssen die einzelnen Elemente in das Dokument eingetragen werden. Zum Einfügen eines Elements wird mit der Methode *CreateElement* ein neues Element erzeugt. Mit der Eigenschaft *Text* kann der Wert des Elements gesetzt werden. Mit der Methode *appendChild* wird das Element an der entsprechenden Stelle in der Baumstruktur eingehängt. Der folgende Programmausschnitt zeigt das Erstellen des Elements *PC-Grunddaten.*

```
Set oRoot = oDOM.createElement("Inventory")
   oDOM.appendChild oRoot

      Set oElemData = oDOM.createElement("PC-Grunddaten")
      oRoot.appendChild oElemData

         Set oElemVN = oDOM.createElement("Computername")
         oElemData.appendChild oElemVN
         oElemVN.Text = strComputername
         …
         Set oElemVN = oDOM.createElement("Datum_Uhrzeit")
         oElemData.appendChild oElemVN
         oElemVN.Text = CStr(Now)
```

Abschließend muss das XML-Dokument, das bis jetzt im Hauptspeicher aufgebaut wurde, ins Dateisystem gespeichert werden. Hierzu muss lediglich die Methode *Save* mit dem Dateinamen aufgerufen werden.

```
Call oDOM.save(strDateiName)
```

5.3.4 Datenübertragung mittels Winsock

Die abgespeicherte XML-Datei muss anschließend per Winsock auf einen der konfigurierten Anwendungsserver übertragen werden. Die Daten der Anwendungsserver werden in einer Konfigurationsdatei (pcscan.ini) abgelegt. Abbildung 42 zeigt die Konfigurationsdatei. Dabei ist zu sehen, dass zwei Anwendungsserver konfiguriert sind. Die Anwendung ist so ausgelegt dass die erste Kombination von Anwendungsserver / Anwendungsport ausgelesen wird. Danach wird versucht eine Winsockverbindung aufzubauen. Wenn die Verbindung zustande kommt werden die Daten übertragen ansonsten wird die nächste Kombination ausgelesen.

```
[Parameter]
Ausgabepfad=c:\programme\noel
```

```
DeleteXML=1
[Anwendungsserver]
AnwServ1=NTPLHW13
AnwPort1=42345
AnwServ2=NTPRZW13
AnwPort2=42345
```

Abbildung 42: Konfigurationsdatei pcscan.ini

Für die Übertragung mittels Winsock stellt Visual Basic ein Steuerelement vom Typ-Winsock zur Verfügung. Bei der Verwendung muss ein entsprechender Mechanismus implementiert werden, der nach dem Versuch des Verbindungsaufbaues abwartet ob eine Antwort zurückkommt ohne den PC zu belasten. In der Praxis hat sich eine Dauer von 5 Sekunden bewährt um Abzuwarten und danach den nächsten Anwendungsserver zu kontaktieren. Nachfolgender Programmausschnitt zeigt die Schleife über die konfigurierten Anwendungsserver und die Implementierung für das Warten auf eine Antwort.

```
i = 1
strAnwendungsserver = fnLeseIni("Anwendungsserver", "AnwServ" & _
   CStr(i), strINIDatei)
strAnwendungsport = fnLeseIni("Anwendungsserver", "AnwPort" & _
   CStr(i), strINIDatei)
Do While strAnwendungsserver <> ""
   With Winsock1
     .RemotePort = CLng(strAnwendungsport)
     .RemoteHost = strAnwendungsserver
     .Connect
   End With
   timeout = DateAdd("s", 5, Time())'5 Sekunden auf Antwort warten
   Do
      DoEvents
   Loop Until Time() > timeout Or Winsock1.State = sckConnected
   If Winsock1.State = sckConnected Then
      fnWinsockSendBinaryFile strDatei
         Exit Do
   End If
   'nächsten Anwendungsserver/Anwendungsport auslesen


   Winsock1.Close
   i = i + 1
```

```
   strAnwendungsserver = fnLeseIni("Anwendungsserver", "AnwServ" & _
      CStr(i), strINIDatei)
   strAnwendungsport = fnLeseIni("Anwendungsserver", "AnwPort" & _
      CStr(i), strINIDatei)
Loop
```

Die eigentliche Datenübertragung erledigt die Funktion fnWinsocksendBinaryFile. Der Funktion wird der Dateiname mit Pfadangabe übergeben. Der Datenstrom wird segmentiert und Blockweise zum Server übertragen. Zwischen den Übertragungen der einzelnen Blöcke wird mit den Befehl *DoEvents* die Steuerung ans Betriebssystem zurückgegeben. Damit wird verhindert dass bei langsameren Clients die CPU-Auslastung sehr hohe Werte erreicht.

```
Const BlockSize = 8192
With Winsock1
   'Header für die Gegenstelle
   .SendData "<begin Size=" & CStr(lngFileSize) & ";" & strFile & ">"
   'XML-Datei als Datenstrom schicken
   Do While lngFilePos < lngFileSize
      lngBytesToRead = BlockSize
      If lngFilePos + lngBytesToRead > lngFileSize Then
         lngBytesToRead = lngFileSize - lngFilePos
      End If
      strBuffer = Space$(lngBytesToRead)
      Get #f, , strBuffer
      .SendData strBuffer
      lngFilePos = lngFilePos + lngBytesToRead
      DoEvents              'Steuerung ans Betriebssystem abgeben
   Loop
...
```

5.4 Serveranwendung

5.4.1 Empfang und Abspeichern der XML-Dateien

5.4.1.1 Hauptprogramm

Zum Abspeichern der XML-Datei wurde ein TCP-Server implementiert. Als Erstes wird ein Listener erzeugt der auf die eingehenden Clientverbindungen horcht. Das Listener Objekt stellt der Namensraum System.Net.Sockets zur Verfügung. Da der TCP-Server die Anfragen von vielen Clients gleichzeitig verarbeiten soll wird pro eingehender Verbindungsanforderung ein Thread der Klasse *ClientHandle*r erzeugt. Der Klasse

Clienthandler wird das für die Clientanforderung erzeugte Listener Objekt mitgegeben. Dadurch können die Datenübertragungen der Clients quasi parallel abgearbeitet werden. Im Test wurden 500 Clientverbindungen gleichzeitig gestartet und die Verarbeitung konnte für alle Clients durchgeführt werden. Zurzeit ist die Anwendung als Konsolenanwendung erzeugt worden. Da aber alle Ausgaben (Starten der Anwendung, Fehlermeldungen) über die Klasse *Log* wahlweise auch in eine Datei geschrieben werden können, wäre als nächster Schritt die Implementierung als Windows Service möglich.

```
// Erstellen eines neuen Listeners  an Anschluss <Port>.
TcpListener listener = new TcpListener(IPAddress.Parse("127.0.0.1"),
    Port);

Log.WriteLine("Server: Initialisiere Anschluss.");
listener.Start();
Log.WriteLine("Server: Warte auf Verbindungsanforderungen");

int clientNum = 0;
while (true)
{
    try
    {
       //Auf eingehende Verbindungsanforderung warten und einen für
       //die Kommunikation initalisierten TcpClient zurückgeben.
        TcpClient client = listener.AcceptTcpClient();
        Log.WriteLine("Server: Verbindung akzeptiert.");
        // Ein neues Objekt für Verbindungsanforderung erstellen.
        clientNum++;
        ClientHandler handler = new ClientHandler(client, "Client " +
            clientNum.ToString());
        // Dieses Objekt in einem anderen Thread ausführen.
        Thread handlerThread =
            new Thread(new ThreadStart(handler.Start));
        handlerThread.IsBackground = true;
        handlerThread.Start();

    }
    catch (Exception err)
    {
        Log.WriteLine(err.ToString());
    }
}
```

5.4.1.2 Die Klasse ClientHandler

Die Klasse *ClientHandler* empfängt den Datenstrom und speichert diesen in einer Datei im lokalen Dateisystem ab. Vom Client wird ein Header gesendet, in welcher der Dateiname für die Speicherung mit gesendet wird.

Diese Startkodierung ist wie folgt aufgebaut:

```
"<begin size=132462;L008210B.XML>"
```

Es werden dabei die Anzahl der Bytes der Datei und der Dateiname übermittelt. Der Programmausschnitt zeigt das Gerüst der Definition der Klasse *ClientHandler*.

```
class ClientHandler
{
    private TcpClient client;
    private string ID;
    public ClientHandler(TcpClient client, string ID)
    {
        this.client = client;
        this.ID = ID;
    }

    public void Start()
    {
        // Den Netzwerk-Stream abrufen.
        NetworkStream stream = client.GetStream();
        StreamWriter myFile;
        LogWriter Log = new LogWriter(strLogdatei);
        // Einen BinaryReader erstellen, um aus dem Stream zu lesen.
        BinaryReader r = new BinaryReader(stream,
           System.Text.Encoding.UTF8);
        string strBuffer;
        char[] Chars;
        Chars = r.ReadChars(8192);
        strBuffer = new String(Chars);
        try
        {
            if (strBuffer.StartsWith("<begin"))
            {
                // Datei erzeugen
                int intIndex1 = strBuffer.IndexOf(";");
                int intIndex2 = strBuffer.IndexOf("<?");Datei
                string strDateiname = strVerz +
                       strBuffer.Substring(intIndex1 + 1,
                       intIndex2 - (intIndex1 + 2));
                strBuffer = strBuffer.Remove(0, intIndex2);
                myFile = new StreamWriter(strDateiname, false);
                while (strBuffer != "")
                {
                    if (strBuffer.EndsWith("/Inventory"))
                    {
                        //Verarbeitung und Datei schliessen
                        myFile.Write(strBuffer);
                        myFile.Close();
                    }
                    else
                    {
                        //Verarbeitung der Daten ohne Endebehandlung
```

```
                        myFile.Write(strBuffer);
                    }
                    strBuffer = "";
                    try
                    {
                        Chars = r.ReadChars(8192);
                        strBuffer = new String(Chars);
                    }
                    catch
                    {
    ...
```

5.4.2 Verarbeitung der XML-Dateien

Für die Verabeitung der XML-Dateien wurde eine Konsolenanwendung entwickelt die über die „Geplanten Tasks“ des Betriebssystems gestartet wird. Für die Entwicklung war es vorrangig, dass die Verarbeitung so rasch stattfindet, dass das Programm in einem Stunden Rhythmus gestartet werden kann. Von Vorteil ist, dass durch das Clientscanprogramm die Datenqualität vorgegeben ist, so dass komplexe Prüfungen der Eingangsdaten nicht notwendig sind.

5.4.2.1 Einlesen der XML-Daten

Für das Behandeln von XML-Dateien steht der Namensraum System.XML im .Net Framework zur Verfügung. Der nachfolgende Programmausschnitt zeigt die Hauptschleifen zum Verarbeiten der XML-Dateien und das Auslesen der Inhalte.

```
...
// Zugriff auf das Dateiverzeichnis
DirectoryInfo d = new DirectoryInfo(strVerzeichnis);
if (!d.Exists)
{
   Log.WriteLine("Verzeichnis zu XML-Dateien nicht gefunden!");
   return;
}
foreach (FileInfo f in d.GetFiles("*.xml"))
{
   XmlDocument doc = new XmlDocument();
   doc.Load(f.DirectoryName + "\\" + f.Name);
   XmlElement root = doc.DocumentElement;

   //Element PC-Grunddaten in XML-Node lesen
   XmlNode pcDaten;
   pcDaten = root.SelectSingleNode("PC-Grunddaten");

   ...

   //Sequence software-liste einlesen
```

```
   XmlNode swListe;
   swListe = root.SelectSingleNode("software-liste");
   // Schleife über alle Softwareeinträge
   foreach (XmlNode sw in swListe.ChildNodes)
...
```

Der Zugriff auf die einzelnen Werte kann mit nachfolgenden Anweisungen erfolgen:

```
System.DateTime dateTime =
   System.DateTime.Parse(pcDaten["Datum_Uhrzeit"].InnerText);
string strComputername = pcDaten["Computername"].InnerText;
string strDisplayVersion = sw["DisplayName"].InnerText;
```

Das Einpflegen der Daten in die Datenbank erfolgt über gekapselte Zugriffsklassen für jede Tabelle. Der Zugriff selbst erfolgt über ADO.NET. Nachfolgender Programmausschnitt zeigt den Zugriff auf die Tabelle *tClient*.

```
// Zugriff auf Tabelle TClient, Datenbank öffnen
TClientBusiness tclientbusiness = new TClientBusiness();
TClients tclients = new TClients();
TClient tclient = new TClient();
...
tclients =
tclientbusiness.GetByComputername(pcDaten["Computername"].InnerText);
if (tclients.Count != 0){
  tclient = tclients.First();
  if (tclient != null){
    System.DateTime dateTime =
       System.DateTime.Parse(pcDaten["Datum_Uhrzeit"].InnerText);
    tclient.Scandatum = dateTime;
    tclient.IPAdresse = pcDaten["IP-Adresse"].InnerText;
    tclient.MACAdresse = pcDaten["MAC-Adresse"].InnerText;
    tclientbusiness.Save(tclient);
...
```

6 Zusammenfassung der Ergebnisse und Ausblick

Es folgt zuerst ein Vergleich mit den im Kapitel 3 ermittelten Anforderungen. Dadurch lässt sich abschätzen, inwieweit die Umsetzung den Vorgaben entspricht. Abschließend werden die Erkenntnisse, die aus der prototypischen Umsetzung gewonnen wurden, dargelegt.

6.1 Ergebnisse

Es werden hier die Anforderungen dem Ist-Zustand des realisierten Prototyps gegenübergestellt. Das primäre Ziel, die Erweiterung des bestehenden Informationssystems RW-Assetmanagement zu einer SAM-Lösung wurde erreicht. In einem ersten Probebetrieb wurden die Daten der Clients der NÖL gescannt und in der Datenbank abgelegt. Die statistischen Kennzahlen aus dem Probebetrieb werden in Tabelle 3 dargestellt.

Kennzahl	Wert
Anzahl der gescannten Clients	8.015
Anzahl der erfassten Benutzer	7.049
Anzahl der gefunden Softwareinstallation	7.393
Anzahl der Softwareinstallationen die als Standard definiert wurden	2.860
durchschnittliche Anzahl der Softwareinstallationen je Client zusätzlich zum Standard	~ 6
Durchschnittliche Verarbeitungsdauer eines Imports	18 Minuten
Datenbankgröße	~ 800 MByte

Tabelle 3: Kennzahlen SAM-Lösung

6.1.1 Funktionale Anforderungsmerkmale

6.1.1.1 Clientanwendung

Das Rollenkonzept wurde zu 100 Prozent in der Clientanwendung realisiert. Bei der Visualisierung der Daten werden die Softwareinstallationen angezeigt, die nicht dem NÖL-Standard entsprechen. Wenn die Verknüpfung zum RW-Assetmanagement hergestellt wurde, kann der entsprechende Eintrag im RW-Assetmanagement direkt angezeigt werden. Auf die Anzeige der lokal abgelegten Daten wurde verzichtet, da diese im XML-Format vorliegen, und in einem Browser ansprechend dargestellt werden.

6.1.1.2 Clientscananwendung

Die Clientscananwendung kann die spezifizierten Daten auslesen und an die Serveranwendung senden. Die ausgelesenen Daten werden, je nach Konfiguration, im lokalen Dateisystem abgelegt. Durch die offene Struktur der Übergabeschnittstelle (XML-Datei) können Erweiterungen mit geringem Aufwand umgesetzt werden.

6.1.1.3 Serveranwendung

Die Leistungsfähigkeit wurde im Probetrieb verifiziert. Der Empfang der Daten und das Abspeichern im lokalen Dateisystem von den rund 8000 Clients konnte mit zwei installierten Anwendungsservices für den Empfang problemlos bewältigt werden. Bei Konfiguration eines einzigen Anwendungsservices traten Engpässe auf, so dass circa ein Viertel der Clients die Daten nicht abliefern konnten.

Der Datenimport, welcher im Stundenintervall gestartet wird, dauert im Schnitt 18 Minuten. Somit stehen die Daten stundenaktuell zur Verfügung. Die empfangenen Dateien werden nach der Verarbeitung verschoben. Somit steht der aktuelle Stand auch in Form der XML-Dateien zur Verfügung.

Auswertungen werden derzeit, auf Anforderung, mit dem in der NÖL eingesetzten Softwareprodukt InfoZoom erledigt. Hier zeigt sich ein Nachteil zu einer Standardlösung welche Berichte in vielfältiger Form bereits beinhalten.

6.1.2 Systembezogene Anforderungsmerkmale

Die systembezogenen Anforderungsmerkmale wurden erfüllt. Auf Seite des Client wird das Betriebssystem Windows XP unterstützt. Die Programme am Server laufen unter Windows Server 2008. Als Datenbanksystem wurde Microsoft SQL Server 2008, wie definiert, eingesetzt.

Bei der Programmierung wurden ebenfalls die geforderten Entwicklungsumgebungen eingesetzt. Insbesondere wurde auf WPF eingegangen und die Tauglichkeit für datenorientierte Anwendungen evaluiert, auch wenn in der gegenständlichen Umsetzung noch keine großen Vorteile gesehen wurden.

Als besonders wichtig wurde die Einbettung in die bestehende Infrastruktur gesehen. Die entwickelten Komponenten konnten allesamt auf der bestehenden Infrastruktur implementiert werden. Hier zeigt sich naturgemäß ein Vorteil der Eigenentwicklung.

6.1.3 Qualitätsbezogene Anforderungsmerkmale

Dadurch, dass eine Eigenentwicklung durchgeführt wurde, konnte die bestehende Infrastruktur optimal genutzt werden. Durch die Einpflege von Schlüsselwerten in die SAM-Datenbank (*LinktoRemedy, LinktoLakis)* können Funktionen zur automatischen Anzeige von Daten in den Zielsystemen genutzt werden.

Eine besondere Herausforderung war die Lösung des Berechtigungssystems, da es sich weitgehend automatisch ergeben sollte. Mit Hilfe der Daten, die am Client ausgelesen werden, bildet sich auch automatisch das Berechtigungssytem ab. Es müssen lediglich jene Clients überarbeitet werden auf denen mehrere Benutzer regelmäßig einsteigen.

Bei der Umsetzung des Datenimportes wurden die Zugriffe auf die Datenbank mit ADO.NET umgesetzt. Zurzeit sind hier keine Engpässe zu beobachten. Bei Weiterentwicklung des Systems und Aufnahme zunehmender Informationen sollte dies speziell beobachtet werden.

6.2 Ausblick

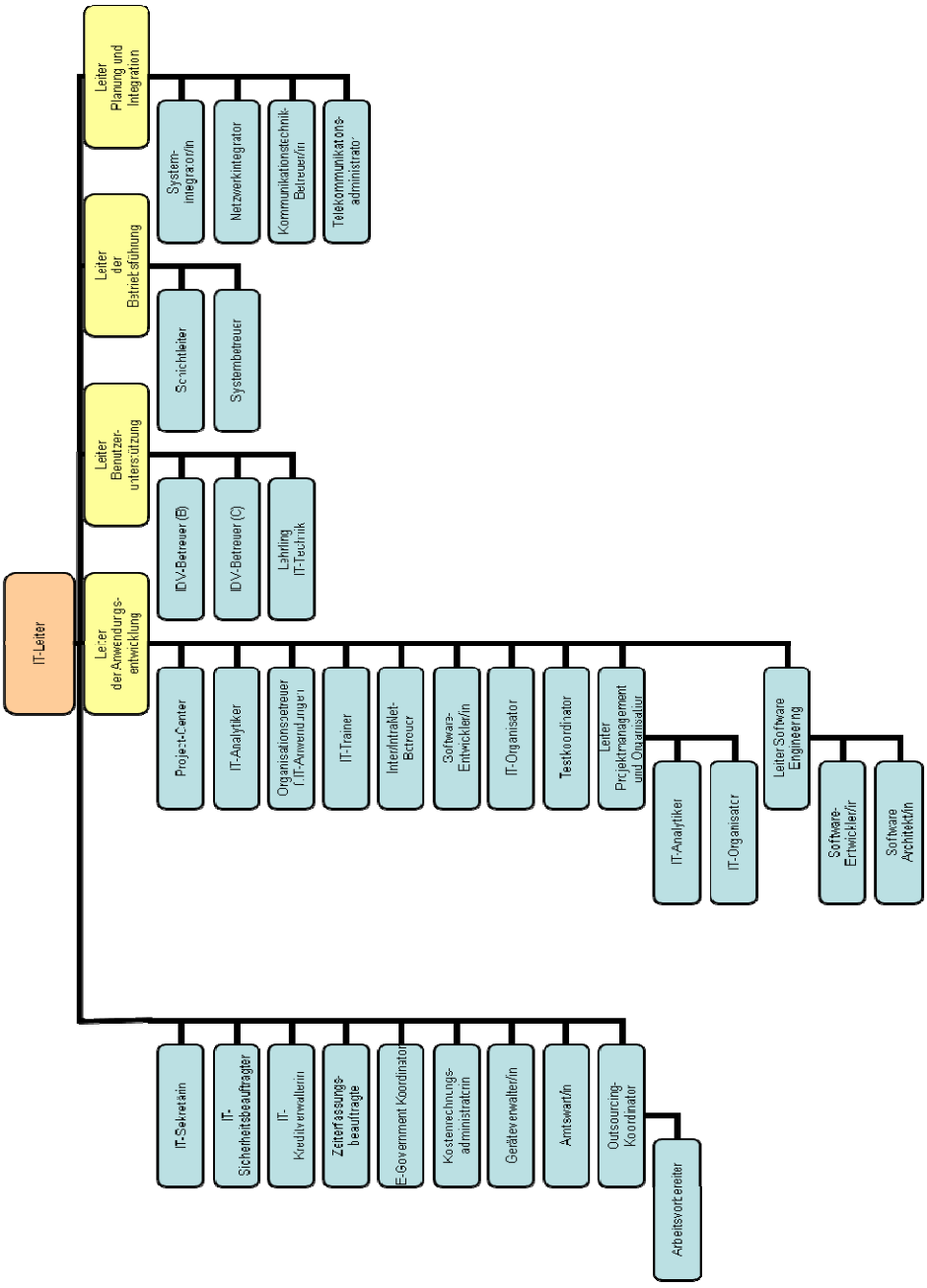
Der Prototyp hat gezeigt, dass es möglich und sinnvoll ist, eine bestehende Inventarisierungsslösung zu einer vollständigen SAM-Lösung, die den Standards entspricht, auszubauen.

Der Prototyp wurde von Anfang an mit besonderem Augenmerk auf eine leichte Erweiterbarkeit entwickelt. Es existieren ein sauberes Konzept und Design, die Entwicklung basiert auf verbreiteten Standards und die Umsetzung ist gut dokumentiert.

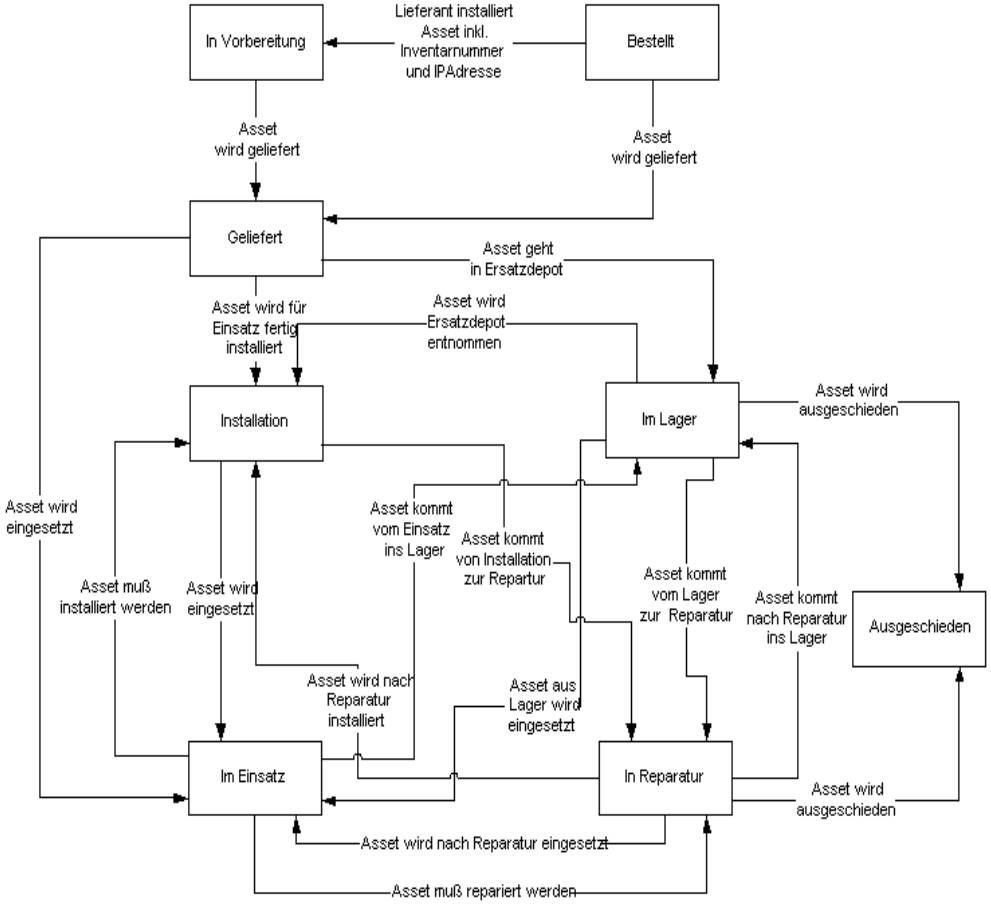
Durch die Entwicklung einer eigenen SAM-Lösung hat die NÖL die Möglichkeit, kostengünstig sowie vor allem schnell und flexibel auf die zukünftigen Anforderungen zu reagieren. Es sollte aber nicht unerwähnt bleiben, dass in gewissen Bereichen noch einiger Aufwand für eine ganzheitliche Lösung zu investieren ist. Vor allem muss das Berichtswesen implementiert und in der NÖL organisatorisch etabliert werden. Erst mit diesem letzten Schritt ist ein sichtbarer Nutzen für einzelne Abteilungen zu sehen.

Anlagen

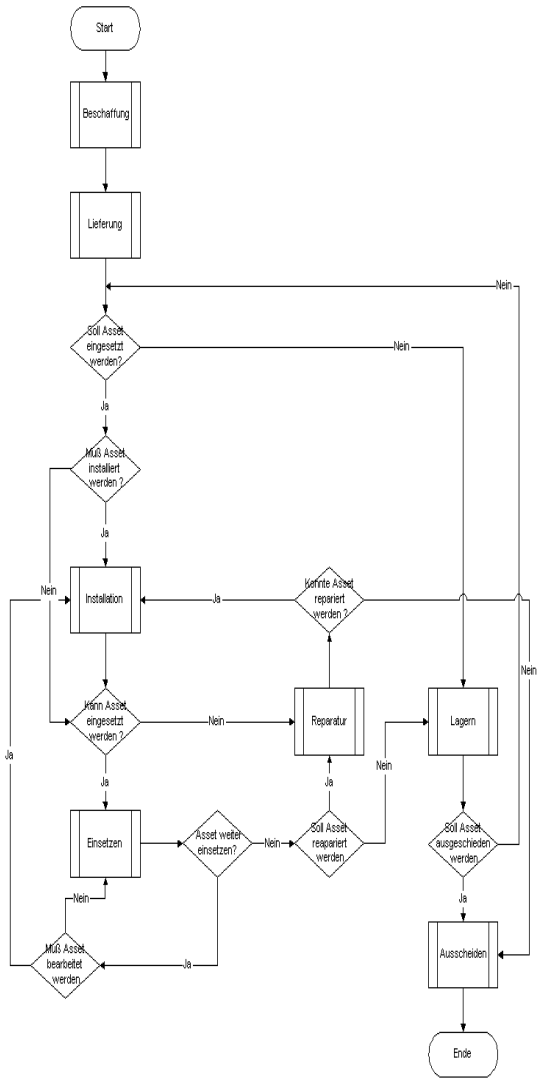
Anlage 1: Organigramm LAD1-IT



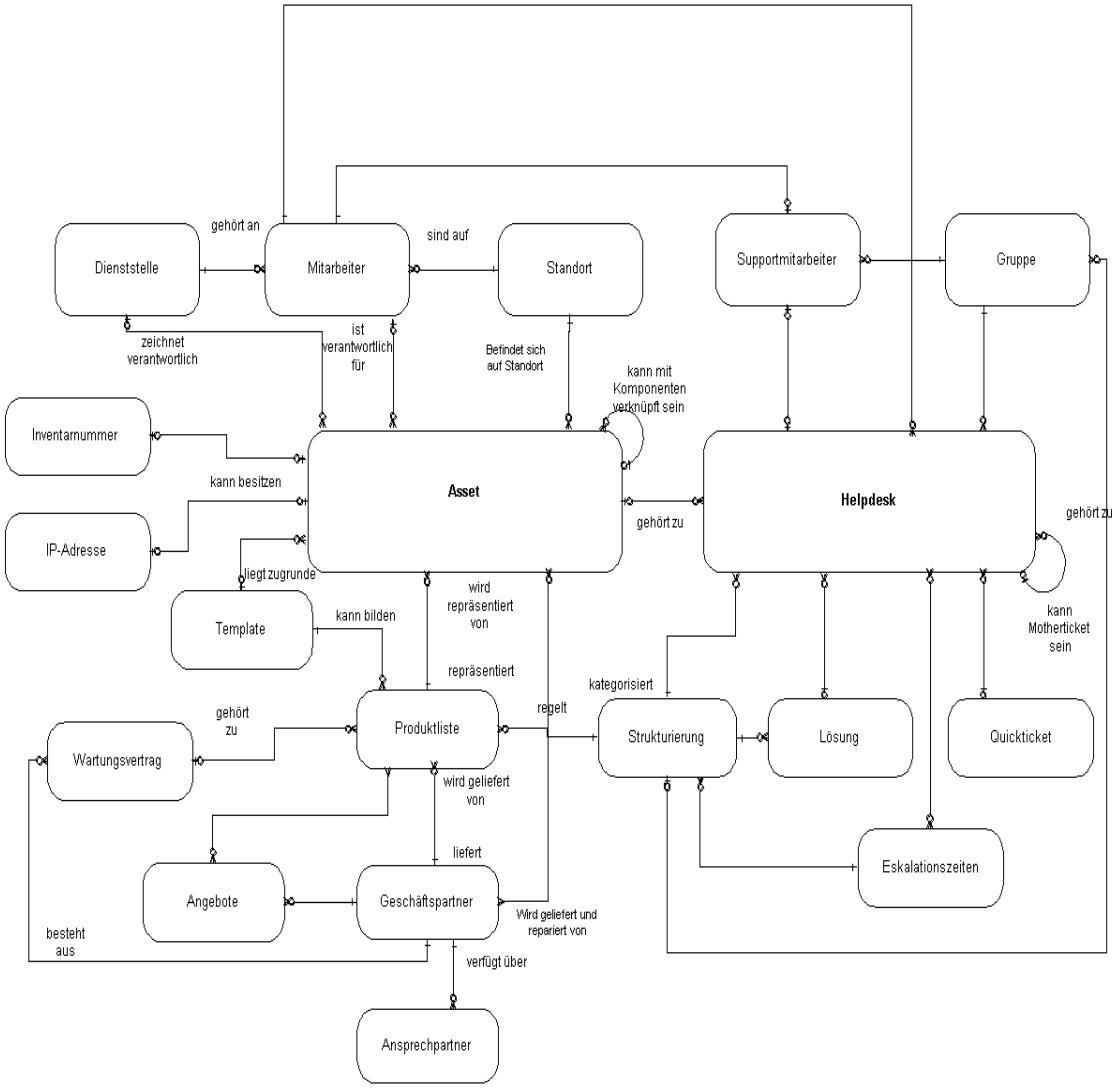
Anlage 2: Asset Lebenszyklus



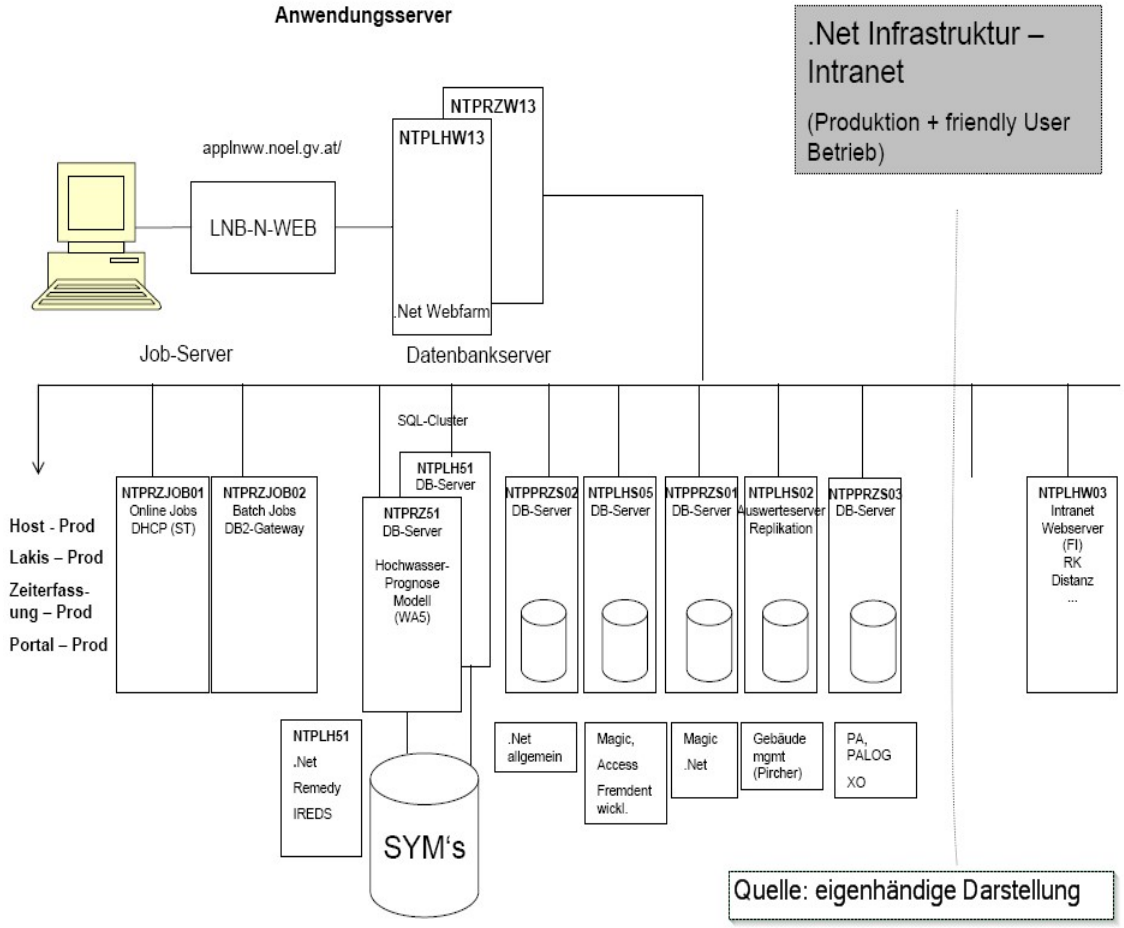
Anlage 3: Asset-Management-Prozess im Überblick

Workflow	**Beschreibung**
	Beschaffungsprozeß wird durch eine Anforderung aus dem LAKIS oder durch eine Problemmeldung ausgelöst. Dem Lieferanten können vor Lieferung bereits Inventarnummer, Computername und IP-Adresse bekanntgegeben werden, damit er die Geräte dementsprechend vorbereiten kann. Das bestellte Asset wird geliefert. Das Asset kann eingesetzt werden. Es ist somit einem Arbeitsplatz und ggf. einem Mitarbeiter zugeordnet. Das Asset kann eingesetzt werden, zum Einsatz vorbereitet werden oder gelagert werden. Bei Defekt kann das Asset zur Reparatur eingeschickt oder über das Lager ausgeschieden werden.

Anlage 4: ER-Modell RW-Assetmanagement



Anlage 5: Infrastruktur für datenorientierte Anwendungen



Anlage 6: Datenbankdefinition in DDL

Tabelle tAbteilung

```
CREATE TABLE [dbo].[tAbteilung] (
	[IDtAbteilung] [int] IDENTITY (1, 1) NOT NULL ,
	[Bezeichnung] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[IT-Koord] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[E-Mail] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[tAbteilung] ADD
	CONSTRAINT [aaaaatAbteilung_PK] PRIMARY KEY  NONCLUSTERED
	(
		[IDtAbteilung]
	)  ON [PRIMARY]
GO

 CREATE  UNIQUE  INDEX [Bezeichnung] ON [dbo].[tAbteilung]([Bezeichnung]) ON [PRIMARY]
GO
```

Tabelle tLizenzmodell

```
CREATE TABLE [dbo].[tLizenzmodell] (
	[IDtLizenzmodell] [int] IDENTITY (1, 1) NOT NULL ,
	[Bezeichnung] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Vertrag] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Bemerkung] [ntext] COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Anzahl_Lizenzen] [int] NULL ,
	[Basis] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[erstellt_am] [datetime] NULL ,
	[upsize_ts] [timestamp] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[tLizenzmodell] ADD
	CONSTRAINT [DF__tLizenzmo__Anzah__7B905C75] DEFAULT (0) FOR [Anzahl_Lizenzen],
	CONSTRAINT [aaaaatLizenzmodell_PK] PRIMARY KEY  NONCLUSTERED
	(
		[IDtLizenzmodell]
	)  ON [PRIMARY]
GO

 CREATE  INDEX [IDtLizenzmodell] ON [dbo].[tLizenzmodell]([IDtLizenzmodell]) ON [PRIMARY]
GO
```

Tabelle tBenutzer

```
CREATE TABLE [dbo].[tBenutzer] (
	[IDtBenutzer] [int] IDENTITY (1, 1) NOT NULL ,
```

```
	[UserId] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Name] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Manger] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[IDtAbteilung] [int] NULL ,
	[eingestiegen_am] [datetime] NULL ,
	[erstellt_am] [datetime] NULL
) ON [PRIMARY]
GO
```

Tabelle tClient

```
CREATE TABLE [dbo].[tClient] (
	[IDtCLient] [int] IDENTITY (1, 1) NOT NULL ,
	[Computername] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[IPAdresse] [nvarchar] (15) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[MACAdresse] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Scandatum] [datetime] NULL ,
	[IDtAbteilung] [int] NULL ,
	[erstellt_am] [datetime] NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[tBenutzer] ADD
	CONSTRAINT [DF__tBenutzer__IDtAb__0CBAE877] DEFAULT (0) FOR [IDtAbteilung],
	CONSTRAINT [aaaaatBenutzer_PK] PRIMARY KEY  NONCLUSTERED
	(
		[IDtBenutzer]
	)  ON [PRIMARY]
GO

CREATE  INDEX [IDtAbteilung] ON [dbo].[tBenutzer]([IDtAbteilung]) ON [PRIMARY]
GO

CREATE  INDEX [IDtBenutzer] ON [dbo].[tBenutzer]([IDtBenutzer]) ON [PRIMARY]
GO

CREATE  INDEX [tAbteilungtBenutzer] ON [dbo].[tBenutzer]([IDtAbteilung]) ON [PRIMARY]
GO

CREATE  UNIQUE  INDEX [UserId] ON [dbo].[tBenutzer]([UserId]) ON [PRIMARY]
GO
```

Tabelle tSoftware

```
CREATE TABLE [dbo].[tSoftware] (
	[IDtSoftware] [int] IDENTITY (1, 1) NOT NULL ,
	[BezeichnungIntern] [nvarchar] (22) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Bezeichnung] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Displayversion] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Bemerkung] [ntext] COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Kategorie] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
```

```
	[Standard] [bit] NOT NULL ,
	[LinktoRemedy] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[LizenzRequired] [bit] NOT NULL ,
	[IDtLizenzmodell] [int] NULL ,
	[erstellt_am] [datetime] NULL ,
	[upsize_ts] [timestamp] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[tSoftware] ADD
	CONSTRAINT [DF__tSoftware__Stand__76CBA758] DEFAULT (0) FOR [Standard],
	CONSTRAINT [DF__tSoftware__Lizen__77BFCB91] DEFAULT (0) FOR [LizenzRequired],
	CONSTRAINT [DF__tSoftware__IDtLi__78B3EFCA] DEFAULT (0) FOR [IDtLizenzmodell],
	CONSTRAINT [aaaaatSoftware_PK] PRIMARY KEY  NONCLUSTERED
	(
		[IDtSoftware]
	)  ON [PRIMARY]
GO

 CREATE  UNIQUE  INDEX [BezeichnungIntern] ON [dbo].[tSoftware]([BezeichnungIntern]) ON [PRI-
MARY]
GO

 CREATE  UNIQUE  INDEX [Displayversion] ON [dbo].[tSoftware]([Displayversion]) ON [PRIMARY]
GO

 CREATE  INDEX [IDtLizenzmodell] ON [dbo].[tSoftware]([IDtLizenzmodell]) ON [PRIMARY]
GO

 CREATE  INDEX [IDtSoftware] ON [dbo].[tSoftware]([IDtSoftware]) ON [PRIMARY]
GO

 CREATE  INDEX [Kategorie] ON [dbo].[tSoftware]([Kategorie]) ON [PRIMARY]
GO

 CREATE  INDEX [tLizenzmodelltSoftware] ON [dbo].[tSoftware]([IDtLizenzmodell]) ON [PRIMARY]
GO
```

Tabelle tBenutzer_Software

```
CREATE TABLE [dbo].[tBenutzer_Software] (
	[IDtBenutzer_Software] [int] IDENTITY (1, 1) NOT NULL ,
	[zugeordnet_am] [datetime] NULL ,
	[LinktoLakis] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[Bemerkung] [ntext] COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[IDtBenutzer] [int] NULL ,
	[IDtSoftware] [int] NULL ,
	[erstellt_am] [datetime] NULL ,
	[upsize_ts] [timestamp] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
```

```
GO

ALTER TABLE [dbo].[tBenutzer_Software] ADD
	CONSTRAINT [DF__tBenutzer__IDtBe__08EA5793] DEFAULT (0) FOR [IDtBenutzer],
	CONSTRAINT [DF__tBenutzer__IDtSo__09DE7BCC] DEFAULT (0) FOR [IDtSoftware],
	CONSTRAINT [aaaaatBenutzer_Software_PK] PRIMARY KEY  NONCLUSTERED
	(
		[IDtBenutzer_Software]
	)  ON [PRIMARY]
GO

 CREATE  INDEX [IDtBenutzer] ON [dbo].[tBenutzer_Software]([IDtBenutzer]) ON [PRIMARY]
GO

 CREATE  INDEX [IDtBenutzer_Software] ON [dbo].[tBenutzer_Software]([IDtBenutzer_Software]) ON
[PRIMARY]
GO

 CREATE  INDEX [IDtSoftware] ON [dbo].[tBenutzer_Software]([IDtSoftware]) ON [PRIMARY]
GO

 CREATE  INDEX [tBenutzertBenutzer_Software] ON [dbo].[tBenutzer_Software]([IDtBenutzer]) ON [PRI-
MARY]
GO

 CREATE  INDEX [tSoftwaretBenutzer_Software] ON [dbo].[tBenutzer_Software]([IDtSoftware]) ON [PRI-
MARY]
GO
```

Tabelle tClient_Benutzer

```
CREATE TABLE [dbo].[tClient_Benutzer] (
	[IDClient_Benutzer] [int] IDENTITY (1, 1) NOT NULL ,
	[IDtCLient] [int] NULL ,
	[IDtBenutzer] [int] NULL ,
	[eingestiegen_am] [datetime] NULL ,
	[erstellt_am] [datetime] NULL
) ON [PRIMARY]
GO

ALTER TABLE [dbo].[tClient_Benutzer] ADD
	CONSTRAINT [DF__tClient_B__IDtCL__023D5A04] DEFAULT (0) FOR [IDtCLient],
	CONSTRAINT [DF__tClient_B__IDtBe__03317E3D] DEFAULT (0) FOR [IDtBenutzer],
	CONSTRAINT [aaaaatClient_Benutzer_PK] PRIMARY KEY  NONCLUSTERED
	(
		[IDClient_Benutzer]
	)  ON [PRIMARY]
GO

 CREATE  INDEX [IDClient_Benutzer] ON [dbo].[tClient_Benutzer]([IDClient_Benutzer]) ON [PRIMARY]
GO
```

```
 CREATE  INDEX [IDtBenutzer] ON [dbo].[tClient_Benutzer]([IDtBenutzer]) ON [PRIMARY]
GO

 CREATE  INDEX [IDtCLient] ON [dbo].[tClient_Benutzer]([IDtCLient]) ON [PRIMARY]
GO

 CREATE  INDEX [tBenutzertClient_Benutzer] ON [dbo].[tClient_Benutzer]([IDtBenutzer]) ON [PRIMARY]
GO

 CREATE  INDEX [tClienttClient_Benutzer] ON [dbo].[tClient_Benutzer]([IDtCLient]) ON [PRIMARY]
GO
```

Tabelle tClient_Software

```
CREATE TABLE [dbo].[tClient_Software] (
	[IDtClient_Software] [int] IDENTITY (1, 1) NOT NULL ,
	[InstallDate] [nvarchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[InstallLocation] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[ScanDatum] [datetime] NULL ,
	[Bemerkung] [ntext] COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[LinktoLakis] [nvarchar] (255) COLLATE SQL_Latin1_General_CP1_CI_AS NULL ,
	[IDtCLient] [int] NULL ,
	[IDtSoftware] [int] NULL ,
	[erstellt_am] [datetime] NULL ,
	[upsize_ts] [timestamp] NULL
) ON [PRIMARY] TEXTIMAGE_ON [PRIMARY]
GO

ALTER TABLE [dbo].[tClient_Software] ADD
	CONSTRAINT [DF__tClient_S__IDtCL__7E6CC920] DEFAULT (0) FOR [IDtCLient],
	CONSTRAINT [DF__tClient_S__IDtSo__7F60ED59] DEFAULT (0) FOR [IDtSoftware],
	CONSTRAINT [aaaaatClient_Software_PK] PRIMARY KEY  NONCLUSTERED
	(
		[IDtClient_Software]
	)  ON [PRIMARY]
GO

 CREATE  INDEX [IDtCLient] ON [dbo].[tClient_Software]([IDtCLient]) ON [PRIMARY]
GO

 CREATE  INDEX [IDtClient_Software] ON [dbo].[tClient_Software]([IDtClient_Software]) ON [PRIMARY]
GO

 CREATE  INDEX [IDtSOftware] ON [dbo].[tClient_Software]([IDtSoftware]) ON [PRIMARY]
GO

 CREATE  INDEX [tClienttClient_Software] ON [dbo].[tClient_Software]([IDtCLient]) ON [PRIMARY]
GO

 CREATE  INDEX [tSoftwaretClient_Software] ON [dbo].[tClient_Software]([IDtSoftware]) ON [PRIMARY]
GO
```

Referenzen

```
ALTER TABLE [dbo].[tBenutzer] ADD
	CONSTRAINT [tBenutzer_FK00] FOREIGN KEY
	(
		[IDtAbteilung]
	) REFERENCES [dbo].[tAbteilung] (
		[IDtAbteilung]
	) ON DELETE CASCADE  ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[tClient] ADD
	CONSTRAINT [tClient_FK00] FOREIGN KEY
	(
		[IDtAbteilung]
	) REFERENCES [dbo].[tAbteilung] (
		[IDtAbteilung]
	) ON DELETE CASCADE  ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[tSoftware] ADD
	CONSTRAINT [tSoftware_FK00] FOREIGN KEY
	(
		[IDtLizenzmodell]
	) REFERENCES [dbo].[tLizenzmodell] (
		[IDtLizenzmodell]
	) ON DELETE CASCADE  ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[tBenutzer_Software] ADD
	CONSTRAINT [tBenutzer_Software_FK00] FOREIGN KEY
	(
		[IDtBenutzer]
	) REFERENCES [dbo].[tBenutzer] (
		[IDtBenutzer]
	) ON DELETE CASCADE  ON UPDATE CASCADE ,
	CONSTRAINT [tBenutzer_Software_FK01] FOREIGN KEY
	(
		[IDtSoftware]
	) REFERENCES [dbo].[tSoftware] (
		[IDtSoftware]
	) ON DELETE CASCADE  ON UPDATE CASCADE
GO

ALTER TABLE [dbo].[tClient_Benutzer] ADD
	CONSTRAINT [tClient_Benutzer_FK00] FOREIGN KEY
	(
		[IDtBenutzer]
	) REFERENCES [dbo].[tBenutzer] (
		[IDtBenutzer]
	) ON DELETE CASCADE  ON UPDATE CASCADE ,
	CONSTRAINT [tClient_Benutzer_FK01] FOREIGN KEY
```

```
	(
		[IDtCLient]
	) REFERENCES [dbo].[tClient] (
		[IDtCLient]
	)
GO

ALTER TABLE [dbo].[tClient_Software] ADD
	CONSTRAINT [tClient_Software_FK00] FOREIGN KEY
	(
		[IDtCLient]
	) REFERENCES [dbo].[tClient] (
		[IDtCLient]
	) ON DELETE CASCADE  ON UPDATE CASCADE ,
	CONSTRAINT [tClient_Software_FK01] FOREIGN KEY
	(
		[IDtSoftware]
	) REFERENCES [dbo].[tSoftware] (
		[IDtSoftware]
	) ON DELETE CASCADE  ON UPDATE CASCADE
GO
```

Change Data Capture

```
EXEC sys.sp_cdc_enable_table_change_data_capture
	@sourcename_schema = ´dbo´,
	@source_name = ´tAbteilung´,
	@role_name = ´db_admin´,
	@captured_column_list = [IT-Koord]

EXEC sys.sp_cdc_enable_table_change_data_capture
	@sourcename_schema = ´dbo´,
	@source_name = ´tBenutzer´,
	@role_name = ´db_admin´,
	@captured_column_list = IDtAbteilung

EXEC sys.sp_cdc_enable_table_change_data_capture
	@sourcename_schema = ´dbo´,
	@source_name = ´tClient, IPAdresse´,
	@role_name = ´db_admin´,
	@captured_column_list = IDtAbteilung
```

View vData

```
Create  VIEW dbo.vData
AS
SELECT TOP 100 PERCENT dbo.tSoftware.Bemerkung, dbo.tSoftware.Displayversion,
		dbo.tSoftware.Bezeichnung, dbo.tClient.Computername,
		dbo.tBenutzer.UserId, dbo.tSoftware.Standard, dbo.tAbteilung.[IT-Koord],
		dbo.tClient_Software.erstellt_am, dbo.tClient.Scandatum,
		dbo.tAbteilung.IDtAbteilung, dbo.tClient.IDtCLient, dbo.tBenutzer.IDtBenutzer
```

```
FROM      dbo.tAbteilung INNER JOIN
          dbo.tBenutzer ON dbo.tAbteilung.IDtAbteilung = dbo.tBenutzer.IDtAbteilung INNER JOIN
          dbo.tClient ON dbo.tAbteilung.IDtAbteilung = dbo.tClient.IDtAbteilung INNER JOIN
          dbo.tClient_Benutzer ON dbo.tBenutzer.IDtBenutzer = dbo.tClient_Benutzer.IDtBenutzer AND
          dbo.tClient.Scandatum = dbo.tClient_Benutzer.eingestiegen_am AND
          dbo.tBenutzer.eingestiegen_am = dbo.tClient_Benutzer.eingestiegen_am AND
          dbo.tClient.IDtCLient = dbo.tClient_Benutzer.IDtCLient INNER JOIN
          dbo.tClient_Software ON dbo.tClient.IDtCLient = dbo.tClient_Software.IDtCLient INNER JOIN
          dbo.tSoftware ON dbo.tClient_Software.IDtSoftware = dbo.tSoftware.IDtSoftware
ORDER BY dbo.tAbteilung.[IT-Koord], dbo.tClient.Computername, dbo.tSoftware.Bezeichnung
```

Anlage 7: View-Definitionen für den Zugriff auf RW-Assetmanagement

View NOE_Asser

```
Create  VIEW dbo.NOE_Asset
AS
SELECT dbo.T57.C1 AS xcAsset_ID, dbo.T57.C2 AS xcErsteller, dbo.T57.C3 AS xtErstellt_am,
dbo.T57.C4 AS xcAssigned_to, dbo.T57.C5 AS xcLetzte_Aenderung_durch,
      dbo.T57.C6 AS xtLetzten_Aenderung_am, dbo.T57.C7 AS x7Status, dbo.T57.C8 AS xcWartungs-
vertragsnr, dbo.T57.C112 AS xcAssigneeGroup,
      dbo.T57.C100000005 AS MA_, dbo.T57.C100000018 AS Zimmernummer, dbo.T57.C100000029
AS Standort_, dbo.T57.C536870913 AS Preis_ATS,
      dbo.T57.C536870914 AS Gebaeude, dbo.T57.C536870915 AS Trouble_Ticket_Nr_,
dbo.T57.C536870916 AS Reparaturfirma, dbo.T57.C536870917 AS InstallAuftrag,
      dbo.T57.C536870918 AS Bestelldatum, dbo.T57.C536870919 AS Ersatzgeraet,
dbo.T57.C536870920 AS Preis_EURO, dbo.T57.C536870921 AS Installationsauftrag,
      dbo.T57.C536870922 AS xcMessage, dbo.T57.C536870923 AS Rechnungsdatum_Reparatur,
dbo.T57.C536870924 AS xcChardatum_Liefer,
      dbo.T57.C536870927 AS Lieferant, dbo.T57.C536870928 AS Lieferfrist, dbo.T57.C536870929 AS
Reparaturscheinnummer, dbo.T57.C536870930 AS Verliehen_an,
      dbo.T57.C536870931 AS Hersteller, dbo.T57.C536870932 AS Rueckgabedatum,
dbo.T57.C536870933 AS xAssetkonfiguration, dbo.T57.C536870934 AS xcObjektklasse,
      dbo.T57.C536870935 AS xGarantietausch_fertig, dbo.T57.C536870936 AS Geschwindigkeit,
dbo.T57.C536870937 AS Grafikkarte,
      dbo.T57.C536870938 AS xrTemp_Schaltjahr2, dbo.T57.C536870939 AS xcDummy3,
dbo.T57.C536870940 AS xiTemp_Year2,
      dbo.T57.C536870941 AS xInventarnummer_erforderlich, dbo.T57.C536870942 AS xAnt-
wort_Dialogbox, dbo.T57.C536870943 AS Dst_Kennziffer,
      dbo.T57.C536870945 AS xcDummy1, dbo.T57.C536870946 AS Asset_ist, dbo.T57.C536870947
AS Lieferscheinnummer, dbo.T57.C536870950 AS BootPROM,
      dbo.T57.C536870952 AS xcDummy4, dbo.T57.C536870953 AS Type, dbo.T57.C536870955 AS
xcPfad2, dbo.T57.C536870956 AS Bestellnummer,
      dbo.T57.C536870958 AS xcPfad3, dbo.T57.C536870961 AS xiTemp_Day, dbo.T57.C536870962
AS xiTemp_Month, dbo.T57.C536870965 AS Wartung_,
      dbo.T57.C536870981 AS xiTemp_Year, dbo.T57.C536870983 AS xiTemp_DeltaJahr,
dbo.T57.C536870984 AS Stock, dbo.T57.C536870985 AS xiTemp_Delta_Monat,
      dbo.T57.C536870986 AS Garantiefirma, dbo.T57.C536870988 AS Soundkarte,
dbo.T57.C536870989 AS Dst_Kurzbeschreibung_, dbo.T57.C536870992 AS Garantie_,
      dbo.T57.C536870993 AS xiTemp_Ueberlauf_Monat, dbo.T57.C536870994 AS xcTemp_Datum,
dbo.T57.C536870995 AS xrTemp_Schaltjahr,
      dbo.T57.C536871004 AS Vorname, dbo.T57.C536871007 AS Anzahl_Lizenzen,
dbo.T57.C536871008 AS Ausscheidungsdatum,
      dbo.T57.C536871009 AS Finanzierungsart, dbo.T57.C536871010 AS Anschluss,
dbo.T57.C536871017 AS Kostenstelle, dbo.T57.C536871021 AS Entsorgungsdatum,
      dbo.T57.C536871022 AS Lagerart, dbo.T57.C536871023 AS Entsorgungsfirma,
dbo.T57.C536871024 AS Entsorgungskosten,
      dbo.T57.C536871025 AS Ausscheidungserloes, dbo.T57.C536871026 AS Ausscheidungsart,
dbo.T57.C536871039 AS Wartungsart,
      dbo.T57.C536871041 AS Reparaturfrist, dbo.T57.C536871042 AS Wartungsfirma,
dbo.T57.C536871046 AS Garantieart, dbo.T57.C536871053 AS Anschluesse,
      dbo.T57.C536871054 AS Depotgerät, dbo.T57.C536871056 AS Frequenz, dbo.T57.C536871057
AS WDV_User, dbo.T57.C536871070 AS Reparaturbeschreibung,
```

dbo.T57.C536871076 AS xSoftware_Logbuch, dbo.T57.C536871077 AS Bestellung_storniert, dbo.T57.C536871080 AS Begruendung,
dbo.T57.C536871082 AS Rechnungsanschrift, dbo.T57.C536871089 AS Frametype, dbo.T57.C536871090 AS Anzahl_der_Assets,
dbo.T57.C536871092 AS Wartungsfrist, dbo.T57.C536871093 AS Reparaturende, dbo.T57.C536871154 AS IGLieferant, dbo.T57.C536871155 AS MAC_Adresse,
dbo.T57.C536871156 AS KVOrdnungsnummer, dbo.T57.C536871158 AS RepKosten, dbo.T57.C536871159 AS Reparatur, dbo.T57.C536871160 AS Reparaturbeginn,
dbo.T57.C536871161 AS Kaeufer, dbo.T57.C536871162 AS Rechnungsdatum, dbo.T57.C536871163 AS Aufloesung, dbo.T57.C536871164 AS IP_Adresse,
dbo.T57.C536871165 AS Prozessor, dbo.T57.C536871166 AS Festplatte, dbo.T57.C536871167 AS Lochmaske, dbo.T57.C536871168 AS Computername,
dbo.T57.C536871169 AS Motherboard, dbo.T57.C536871170 AS LAN_Type, dbo.T57.C536871172 AS Roehrentyp, dbo.T57.C536871173 AS Papierhandhabung,
dbo.T57.C536871174 AS Papiergroesse, dbo.T57.C536871175 AS Bus, dbo.T57.C536871177 AS IGFinanzierungsart, dbo.T57.C536871178 AS IGAnschaffungswert,
dbo.T57.C536871181 AS IGIntEinfuhrbewilligung, dbo.T57.C536871182 AS IGAllgemeineAnmerkungen, dbo.T57.C536871183 AS IGVermehrtLandesvermögen,
dbo.T57.C536871184 AS IGBestandswert, dbo.T57.C536871185 AS IGRechnungsjahr, dbo.T57.C536871186 AS IGEingabedatum,
dbo.T57.C536871187 AS IGÜbernahmedatum, dbo.T57.C536871188 AS IGBetrifft, dbo.T57.C536871189 AS IGProjekt, dbo.T57.C536871190 AS IGAusscheidungsart,
dbo.T57.C536871191 AS IGAusscheidungsdatum, dbo.T57.C536871195 AS IGKostenstelle, dbo.T57.C536871196 AS IGinterneAuftragsnummer,
dbo.T57.C536871197 AS IGAuftragsnummer, dbo.T57.C536871198 AS IGAuftragstext, dbo.T57.C536871199 AS IGmonatlicheAbschreibung,
dbo.T57.C536871200 AS IGBestandswertKostenrechnung, dbo.T57.C536871201 AS IGEinsatzdatum, dbo.T57.C536871202 AS IGSAPAbstimmkostenstelle,
dbo.T57.C536871203 AS IGNutzungsdauer, dbo.T57.C536871206 AS IGArt, dbo.T57.C536871209 AS IGKennziffer, dbo.T57.C536871210 AS IGEingabeVerrechnung,
dbo.T57.C536871211 AS IGEingabe_Verrechnung, dbo.T57.C536871212 AS IGSummeAW, dbo.T57.C536871213 AS Gattung, dbo.T57.C536871214 AS IGSummeBW,
dbo.T57.C536871215 AS xZusaetzliche_Informationen, dbo.T57.C536871218 AS Rep__Rechnungsnr_, dbo.T57.C536871219 AS xtTimestamp_AC,
dbo.T57.C536871220 AS xiAenderungscode, dbo.T57.C536871222 AS RechnungWeitergeleitet, dbo.T57.C536871235 AS Laufwerk, dbo.T57.C536871236 AS Seiten_Min,
dbo.T57.C536871237 AS BerechMonate, dbo.T57.C536871238 AS Push_TT, dbo.T57.C536871239 AS InventarnummerBC, dbo.T57.C536871240 AS Anzahl_Etiketten,
dbo.T57.C536871241 AS WertBC, dbo.T57.C536871243 AS Druckerwahl, dbo.T57.C536871250 AS verknuepft_mit_Produktbez_Haup,
dbo.T57.C536871251 AS MASTER_Gehoert_zu_, dbo.T57.C536871268 AS LieferantenNr, dbo.T57.C536871270 AS Anzahl_CPUs,
dbo.T57.C536871285 AS DienststelleAlt, dbo.T57.C536871286 AS RB_keineSteuer, dbo.T57.C536871287 AS InventarnummerStrasse,
dbo.T57.C536871288 AS Poenale, dbo.T57.C536871289 AS Belegnummer, dbo.T57.C536871290 AS Gebarungsfallnummer, dbo.T57.C536871291 AS Rimkennziffer,
dbo.T57.C536871293 AS Ausscheidungsbelegnummer, dbo.T57.C536871294 AS AltGarantieende, dbo.T57.C536871295 AS Abverkaufsbemerkung,
dbo.T57.C536871297 AS kmLeistung, dbo.T57.C536871298 AS Verrechnungsansatz, dbo.T57.C536871299 AS Bereich, dbo.T57.C536871305 AS Hostname,
dbo.T57.C536871307 AS RB_manuell, dbo.T57.C536871309 AS Zinssatz, dbo.T57.C536871310 AS Leasingfirma, dbo.T57.C536871311 AS LeasingfirmaNr,

```
        dbo.T57.C536871312 AS Fahrzeuggruppe, dbo.T57.C536871313 AS RB_neugebraucht,
dbo.T57.C536871315 AS Ausscheidungsart_C,
        dbo.T57.C536871316 AS IGAusscheidungsart_C, dbo.T57.C536871317 AS ID,
dbo.T57.C536871318 AS tempUID, dbo.T57.C536871319 AS Leistungsart,
        dbo.T57.C536871320 AS Leistungsartwert, dbo.T57.C536871321 AS VerrechnungsansatzST,
dbo.T57.C536871322 AS Vertragsbeginn,
        dbo.T57.C536871323 AS Vertragsdauer, dbo.T57.C536871324 AS Vertragsende,
dbo.T57.C536871325 AS Erinnerungsempfaenger,
        dbo.T57.C536871326 AS RB_ErinnerungVersendet, dbo.T57.C536871327 AS Bestellbezug,
dbo.T57.C536871328 AS voraussAusscheidungsdatum,
        dbo.T57.C536871329 AS Ersatzteilrabatte, dbo.T57.C536871330 AS SummeSonderausstattung,
dbo.T57.C536871331 AS PreisInklSonder,
        dbo.T57.C536871333 AS InventarnummerErhaltung, dbo.T57.C536871334 AS Erinnerungszeit-
punkt, dbo.T57.C536871335 AS Zahlungsfreigabe,
        dbo.T57.C536871336 AS Rechnungsbearbeiter, dbo.T57.C536871337 AS Euro_pro_km,
dbo.T57.C536871338 AS Behoerdenrabatt,
        dbo.T57.C536871339 AS Versicherungsmarkierung, dbo.T57.C536871340 AS AAmonatlicheRate,
dbo.T57.C536871341 AS KFZ_Anmeldedatum,
        dbo.T57.C536871342 AS KFZ_Abmeldedatum, dbo.T57.C536871343 AS KFZ_PolKennz,
dbo.T57.C536871344 AS Restwert,
        dbo.T57.C536871349 AS TaetigkeitsberichtJahr, dbo.T57.C536871350 AS Endpreis,
dbo.T57.C536871351 AS TaetigkeitsberichtJahr2,
        dbo.T57.C536871352 AS Standort_IDErhaltung, dbo.T57.C536871353 AS Freigabedatum,
dbo.T57.C536871354 AS RB_Zuschrifterledigt,
        dbo.T57.C536871355 AS RestwertLeasingVon, dbo.T57.C536871356 AS RestwertLeasingBis,
dbo.T57.C536871357 AS IGFinanzierungsartST,
        dbo.T57.C536871358 AS alteStandortID, dbo.T57.C536871359 AS TaetigkeitsberichtJahr3,
dbo.T57.C536871360 AS FMZ, dbo.T57.C536871361 AS ST_Code,
        dbo.T57.C536871362 AS Abschreibungsdauer, dbo.T57.C536871363 AS AuftragsArt,
dbo.T57.C600000010 AS Kategorie, dbo.T57.C600000011 AS Gruppe,
        dbo.T57.C600000012 AS Element, dbo.T57.C600070102 AS Seriennummer, dbo.T57.C600070104
AS Lieferdatum, dbo.T57.C600070114 AS Nachname,
        dbo.T57.C600070131 AS xLog_Buch, dbo.T57.C600070208 AS Produktbezeichnung,
dbo.T57.C600070256 AS Speicher__Mb_, dbo.T57.C600070277 AS Hilfselement1,
        dbo.T57.C600070278 AS Hilfselement2, dbo.T57.C600070401 AS Kostenstelle_User,
dbo.T57.C600070503 AS IGEigentuemerSparte_Bedeutung,
        dbo.T57.C610000061 AS Status, CONVERT(varchar(15), dbo.B57.CO610000119) + ';' + CON-
VERT(varchar(15), dbo.B57.CC610000119)
        + ';' + dbo.B57.C610000119 AS Vorlage, CONVERT(varchar(15), dbo.B57.CO610000121) + ';' +
CONVERT(varchar(15), dbo.B57.CC610000121)
        + ';' + dbo.B57.C610000121 AS Vorlage2, dbo.T57.C536871366 AS Gueltigab
FROM        dbo.T57 LEFT OUTER JOIN
          dbo.B57 ON dbo.T57.C1 = dbo.B57.C1
```

Anlage 8: XAML-Definition des Hauptfensters

```
<Window x:Class="SAMClientAnwendung.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
       Title="SAM-Einhaltung und Verifikation" Height="681.803"
       Width="886.844" ResizeMode="CanResizeWithGrip" MinWidth="700"
       MinHeight="500" >
    <Grid>
        <ListView Margin="20,227,19,65.013" Name="listView1"
         ItemTemplate="{DynamicResource SoftwareTemplate}"
         ItemsSource="{Binding Path=Table}">
            <ListView.Background>
                <LinearGradientBrush>
                    <GradientStop Color="WhiteSmoke" Offset="0"/>
                </LinearGradientBrush>
            </ListView.Background>
            <ListView.View>
                <GridView>
                    <GridViewColumn Header="Bezeichnung"
                DisplayMemberBinding="{Binding Path=Bezeichnung}"/>
                    <GridViewColumn Header="DisplayVersion"
                DisplayMemberBinding="{Binding Path=DisplayVersion}"/>
                    <GridViewColumn Header="erstellt am"
                DisplayMemberBinding="{Binding Path=erstellt_am}"/>
                    <GridViewColumn Header="Lizenz"
                DisplayMemberBinding="{Binding Path=Lizenz}"/>
                    <GridViewColumn Header="Bemerkung"
                DisplayMemberBinding="{Binding Path=Bemerkung}"/>
                </GridView>
            </ListView.View>
        </ListView>
        <Button Height="28" HorizontalAlignment="Right" Margin="0,0,19,20"
Name="btnOk" VerticalAlignment="Bottom" Width="143"
Click="btnOk_Click">Ok</Button>
        <Label Height="22" HorizontalAlignment="Left" Margin="23.338,50,0,0"
Name="label1" VerticalAlignment="Top" Width="96">Computername</Label>
        <TextBox Height="23" Margin="128.359,107.021,0,0"
Background="WhiteSmoke" Name="txtBenutzerkennung" VerticalAlignment="Top"
HorizontalAlignment="Left" Width="150" />
        <Label Height="30" HorizontalAlignment="Left" Margin="20,105.021,0,0"
Name="label2" VerticalAlignment="Top" Width="96">Benutzerkennung</Label>
        <Button Height="23" HorizontalAlignment="Left" Margin="286.724,49,0,0"
Name="btnSearch" VerticalAlignment="Top" Width="52" Click="btnSearch_Click" >
            <Button.BitmapEffectInput>
                <BitmapEffectInput />
            </Button.BitmapEffectInput>
            <Button.BitmapEffect>
                <DropShadowBitmapEffect ShadowDepth="2" />
            </Button.BitmapEffect> Search</Button>
         <ListView Margin="428.419,100,19,0" Name="listView2"
MouseLeftButtonUp="listView2_MouseLeftButtonUp"
          ItemTemplate="{DynamicResource SoftwareTemplate}"
ItemsSource="{Binding Path=Table}" Height="107.175" VerticalAlignment="Top"
SelectedValuePath="Computername">
            <ListView.BitmapEffect>
                <OuterGlowBitmapEffect />
            </ListView.BitmapEffect>
            <ListView.Background>
                <LinearGradientBrush>
                    <GradientStop Color="WhiteSmoke" Offset="0"/>
                </LinearGradientBrush>
            </ListView.Background>
```

```
            <ListView.View>
                <GridView>
                    <GridViewColumn
                        Header="Auswahl Computer"
DisplayMemberBinding="{Binding Path=Computername}">
                    </GridViewColumn>
                    <GridViewColumn
                        Header="zugehöriger Benutzer"
DisplayMemberBinding="{Binding Path=UserID}" >
                    </GridViewColumn>
                    <GridViewColumn
                        Header="Scandatum" DisplayMemberBinding="{Binding
Path=Scandatum}" >
                    </GridViewColumn>
                </GridView>
            </ListView.View>
        </ListView>
        <ComboBox Height="23" HorizontalAlignment="Right" Margin="0,50,19,0"
Name="cboManGruppe"
                  VerticalAlignment="Top" Width="350.07" ItemsSource="{Binding
Path=Table}" DisplayMemberPath="Bezeichnung" SelectedValuePath="IDTAbteilung"
SelectionChanged="cboManGruppe_SelectionChanged">
            <ComboBox.BitmapEffect>
                <OuterGlowBitmapEffect />
            </ComboBox.BitmapEffect>
        </ComboBox>
        <Label Height="30" Margin="428.419,50,375,0" Name="label3"
VerticalAlignment="Top">Abteilung:</Label>
        <TextBox Background="WhiteSmoke" Height="23"
HorizontalAlignment="Left" Margin="128.359,138,0,0" Name="textBox1"
VerticalAlignment="Top" Width="150" />
        <TextBox Background="WhiteSmoke" Height="23"
HorizontalAlignment="Left" Margin="128.359,168,0,0" Name="textBox2"
VerticalAlignment="Top" Width="150" />
        <Label Height="30" HorizontalAlignment="Left" Margin="20,138,0,0"
Name="label4" VerticalAlignment="Top" Width="96">Scandatum</Label>
        <Label Height="30" HorizontalAlignment="Left" Margin="23.338,168,0,0"
Name="label5" VerticalAlignment="Top" Width="96">IP-Adresse</Label>
        <Label Height="28" Margin="19.6,203,258,0" Name="label6"
VerticalAlignment="Top">Software zusätzlich zu Standardkonfiguration
(compliance):</Label>
        <Menu Height="22" HorizontalAlignment="Left" Margin="10,5,0,0"
Name="mnuMenu" VerticalAlignment="Top" Width="268.359"
Background="Transparent">
            <MenuItem Name="Datei" Header="Datei">
                <Separator>
                    <Separator.BitmapEffect>
                        <OuterGlowBitmapEffect />
                    </Separator.BitmapEffect>
                </Separator>
                <MenuItem Header="Schließen"/>
            </MenuItem>
            <MenuItem Header="Hilfe" />
        </Menu>
        <Button Height="28" HorizontalAlignment="Left" Margin="23.338,0,0,20"
Name="btnRW" VerticalAlignment="Bottom" Width="255.021">Zeige Daten aus RW-
Assetmanagement</Button>
        <TextBox Height="25.005" HorizontalAlignment="Left"
Margin="128.359,50.361,0,0" Name="txtComputername" VerticalAlignment="Top"
Width="150" />
```

Literaturverzeichnis

ADO.NET 2010
ADO.NET-Architektur. URL: <http://msdn.microsoft.com/de-de/library/aa719511(VS.71).aspx>, verfügbar am 4.6.2010

Alpar 2000
Alpar, P., Grob, H. L., Weinmann, P., Winter, R. (2000): Anwendungsorientierte Wirtschaftsinformatik: Eine Einführung in die strategische Planung, Entwicklung und Nutzung von Informations- und Kommunikationssystemen. - 2. Aufl. - Vieweg, Braunschweig/
Wiesbaden, 2000

Davis/Olson 1985
Davis, Gordon Bitter; Olson, Margrethe H.: Management Information Systems: Conceptual Foundations, Structure, and Development. 2. Aufl., Mcgraw-Hill College : New York, 1985

Dröge/Ratz 2008
Dröge R., Raatz M.: SQL Server 2008 – Überblick über Konfiguration, Administration, Programmierung. Unterschleißheim: Microsoft Press Deutschland, 2008

Dumke 2003
Dumke, R.: Software Engineering. – 4. Aufl. – Wiesbaden: Friedr. Vieweg & Sohn Verlag/GWV Fachverlage GmbH, 2003

Eller 2008
Eller, Frank: Visual C# 2008 – Grundlagen, Programmiertechniken, Datenbanken. – 1.Aufl. – München: Addision-Wesely Verlag, 2008

Heinrich/Lutz 1990
Heinrich, Lutz J.: Der Prozeß der Systemplanung und entwicklung. In: Kurbel, Karl; Strunz, Horst (Hrsg.): Handbuch Wirtschaftsinformatik. Stuttgart, 1990.

ISO/IEC 19770-2
ISO/IEC 19770-2: Software identification tag. URL: <http://en.wikipedia.org/wiki/ISO_19770#ISO.2FIEC_19770-2:_Software_identification_tag>, verfügbar am 10.7.2010

Kalaher 2010
Kalaher, P. <patrick.kalaher@wipro.com> : Microsoft Asset Inventory Service – A Solution for Software Asset Optimization and License Compliance.
URL:<http://download.microsoft.com/download/8/9/D/

89D847B5-92CB-4BBF-9B07-2E0987D23D70/AIS%20Solution%20White%20Paper%20for%20IT%20Professionals.pdf>, verfügbar am 13.6.2010

Kansy 2008
Kansy, Thorsten: Datenbankprogrammierung mit .Net 3.5 – mehrschichtige Applikationen mit Visual Studio 2008 und MS SQL Server 2008. – 1. Aufl. – München: Carl Hanser Verlag, 2008

Ludewig/Lichter 2010
Ludewig J., Lichter H.: Software Engineering – Grundlagen, Menschen, Prozesse, Techniken. – 2. Aufl. – Heidelberg: dPunkt Verlag GmbH, 2010

Macke 2010
Macke, M.: Identifizieren Sie sich In: iX, Mai 2010

Marco 2006
Marco, C.: Durchblick im Lizenzen-Dschungel bewahren In: Computer im Mittelstand, Oktober 2006

Marple 2010
Miss Marple 2010: Asset Management. URL: < http://www.asset-lizenzmanagement.de/de/main/produkte/miss-marple-2010/miss-marple-enterprise-edition-2010/asset-management.html>, verfügbar am 3.7.2010

Monch 2008
Monch, R.: Die Umsetzung von IT-Governance mit ITIL und COBIT vor dem Hintergrund von Sarabanes-Oxely. München: Grin., 2008

Mosers 2010
Mosers, C.: <moc@zuehlke.com>: Routed Events. URL: <http://www.wpftutorial.net/RoutedEvents.html>, verfügbar am 13.7.2010

Mühsig 2010
Mühsig, R.: HowTo: Windows Presentation Foundation (Einstieg, Infos, Programme, Überblick). URL: <http://code-inside.de/blog/2008/01/30/howto-windows-presentation-foundation-einstieg-infos-programme-berblick/>, verfügbar am 21.7.2010

Nieder 2003
Nieder, Hans C.: ZENworks for Desktops 4. - 1.Aufl. - Bonn: mitp-Verlag, 2003

OGC 2007
OGC: Best Practice for Software Asset Management.- 7. Aufl. – London, 2007

RIS 2010
NÖ: Verordnung über die Geschäftsordnung des Amtes der NÖ Landesregierung. URL: http://www.ris.bka.gv.at/Dokument.wxe?Abfrage=LrNo&Dokumentnummer=LRNI_1976001, verfügbar am14.6.2010

Rudd 2009
Rudd, Colin: ITIL V3 Guide to Software Asset Management. London: The Stationery Office, 2009

Ryan 2010
Ryan: Sockets and C#. URL: <http://weblog.cynosura.eu/post/2009/03/02/Sockets-and-C.aspx>, verfügbar am 12.8.2010

SCCM 2010
Microsoft: Asset Intelligence. URL:<http://www.microsoft.com/germany/systemcenter/sccm/evaluation/assetintelligence.mspx>, verfügbar am16.6.2010

Soltys 2000
Soltys, V.: Anforderungsanalyse Problem- und Assetmanagement, interne Unterlage, St. Pölten 2000

Wegener 2009
Wegener, J.: Windows Presentation Foundation – WPF - Grafische Benutzerschnittstellen mit .NET 3.5. - München: Carl Hanser Verlag, 2009

WIKI01 2010
Chen-Notation: URL. <http://de.wikipedia.org/wiki/Chen-Notation>, verfügbar am 10.6.2010

WIKI02 2010
Entity-Relationship-Modell: URL. <http://de.wikipedia.org/wiki/ER-Modell>, verfügbar am 10.6.2010

WIKI03 2010
Socket (Software). URL: <http://de.wikipedia.org/wiki/Socket_(Software)>, verfügbar am, 12.7.2010

Winsock 2010
Getting Started with Winsock. URL: <http://msdn.microsoft.com/de-de/library/ms738545.aspx>, verfügbar am 4.6.2010

Winter/Aier 2010
Winter, R., Aier S.: Informationsystem-Architektur. URL: <http://www.enzyklopaedie-der-wirtschaftsinformatik.de/wi-enzyklopaedie/lexikon/daten-wissen/Informationsmanagement/Information-/Informationssystem-Architektur>, verfügbar am 13.8.2010

XML 2010
XML Completed Work: URL. <http://www.w3.org/TR/#tr_XML>, verfügbar am 10.7.2010

ZENWorks 2010
Novell ZENWorks Asset Management. URL: <www.novell.com/de-de/products/zenworks/assetmanagement>, verfügbar am 3.7.2010

Erklärung zur selbstständigen Anfertigung der Arbeit

Ich erkläre, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Literatur und Hilfsmittel angefertigt habe.

Bearbeitungsort, Datum Unterschrift